ERDC-TEC CR-00-1

Topographic Engineering Center

US Army Corps
of Engineers®
Engineer Research and
Development Center

RADIUS: Model-Based Optimization

SRI INTERNATIONAL: Pascal V. Fua, Christopher I. Connolly, Aaron J. Heller, Lynn H. Quam, and Thomas M. Strat

June 2000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 2000	Final Technical Report -Sep 1992 - Feb 1997

4. TITLE AND SUBTITLE

RADIUS: Model-Based Optimization Final Report

5. FUNDING NUMBERS

DACA76-92-C-034

6. AUTHORS

Thomas M. Strat, Sr. , Lynn H. Quam, Sr. , Pascal V. Fua, Aaron J. Heller, Christopher I. Connolly

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES

Defense Advanced Research Projects Agency
3701 North Fairfax Dr., Arlington, VA 22203-1714
U.S. Army Topographic Engineering Center
7701 Telegraph Road, Alexandria, VA 22315-3864

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ERDC/TEC CR-00-1

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The construction and use of 3-D models of military and industrial sites will allow revolutionary advances in the speed, confidence, and range of analytical techniques with which an Image Analyst (IA) develops and reports intelligence information. This SRI research project, in support of the RADIUS Program, seeks to increase the speed and accuracy with which site models can be constructed from current imagery by developing a new family of image understanding (IU) techniques, and by developing a novel way for an IA to employ them. This report describes the techniques developed at SRI under the RADIUS project. The techniques described here have been integrated and tested in the RADIUS Common Development Environment.

14. SUBJECT TERMS

Computer vision, aerial image analysis, RADIUS, optimization, and snakes

15. NUMBER OF PAGES

132

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This report describes research preformed on the final year of the project RADIUS: Model-Based Optimization. This research is supported by Defense Advanced Research Projects Agency (DARPA) of the Department of Defense (DoD) and is monitored by the U.S. Army topographic Engineering Center (TEC) under the contract DACA76-92-C-0034. The DARPA Program Manager is Dr. Thomas M. Strat and the TEC Contracts Officer Representative (COR) is Ms. Lauretta E. Williams.

1 INTRODUCTION

The construction and use of 3-D models of military and industrial sites will allow revolutionary advances in the speed, confidence, and range of analytical techniques with which an image analyst (IA) develops and reports intelligence information. This SRI research project, in support of the Research and Development in Image Understanding Systems (RADIUS) Program, seeks to increase the speed and accuracy with which site models can be constructed from current imagery by developing a new family of image understanding (IU) techniques, and by developing a novel way for an IA to employ them.

Our research has proceeded on two fronts simultaneously:

- Extending the generality and power of model-based optimization algorithms (MBO)
- Developing a context-based approach to IA control of IU algorithms

We first present our overall research goals and then discuss our progress toward advancing the state-of-the-art in these two areas. This report also incorporates as appendices (A through E) several technical papers describing our research in detail.

2 RESEARCH GOALS

Model-Supported Exploitation (MSE) is the analysis of imagery (by human or computer) with the aid of 2-D and 3-D models of the scene [1, 2, 3]. Two major scientific problems must be solved for MSE to be a viable concept for use in an operational intelligence setting. A successful MSE system must have:

- An interface that enables the IA to easily specify what he or she wants the machine to do.
- A set of algorithms that enables the machine to perform the tasks posed by the IA.

While the majority of research in IU has been concerned with fully automated algorithms for interpreting images, the Perception Group at SRI has made the design and implementation of semiautomated systems a major goal.

- We have extended the MBO technology to provide the capability of extracting many different object classes under a variety of imaging and scene conditions. Integrated into the

RADIUS Common Development Environment (RCDE), this technology constitutes an operational suite of tools tailored to the needs of the site model builder.

- We are developing a software architecture that automatically chooses IU algorithms and their parameters, given modeling tasks posed by an IA. The centerpiece of the approach is a framework that reasons about the context of the given task to make these choices intelligently.

3 MODEL-BASED OPTIMIZATION

Model-based optimization is a paradigm in which an objective function expresses both geometric and photometric constraints on features of interest. A parametric model of a feature (such as a road, building, or coastline) is extracted from one or more images by adjusting the model's state variables until a minimum value of the objective function is obtained. The optimization procedure yields a description that simultaneously satisfies (or nearly satisfies) all constraints and, as a result, is likely to be a good model of the feature.

Implementation of an MBO algorithm requires the specification of four components:

Objective function: A mathematic function that expresses the preferred geometric and photometric properties to be exhibited by the feature.

Representation: The geometric primitives used to represent the feature, thereby limiting the class of features that can be modeled.

Optimization: The procedure to be employed for finding a configuration of the feature that locally minimizes the objective function.

Initial conditions: The configuration of the feature used as the starting point by the optimization procedure.

Our research addresses all four of these areas, seeking to find instantiations of the MBO paradigm that provide effective means for extracting features of interest to the RADIUS Program.

The deformable models we use here are extensions of traditional snakes [4, 5, 6] that we refer to as *generalized snakes.* They are polygonal curve or planar face objects to which is associated an objective function that combines an image-derived term that measures the fit to the image data and a regularization term that enforces geometric constraints.

In the course of the contract, we have extensively reorganized and extended our MBO package to accommodate more and more complex types of objects. Initially, we could only deal with polygonal curves that could be modeled as a sequential list of vertices. However, many objects that are of importance to a photoanalyst and are supported by RCDE, such as road networks or 3-D extruded objects, do not fit this model. Their topology is that of a network and, to describe them completely, one must supply, not only the list of their vertices, but also a list of "edges" that defines the connectivity of those vertices. In addition, with some of these complex objects, one can also define "faces," that is circular lists of vertices that must be constrained to remain planar.

We have therefore introduced new breeds of snake and can now accommodate the full taxonomy of snakes described by Table 1. These generalized snakes are described in detail in Appendix A.

Table 1: Snake taxonomy. The columns represent different types of snake and the rows different kinds of constraint that can be brought to bear. The table entries are examples of objects that can be modeled using these combinations.

Constraints/Type	Simple curve	Ribbon curve	Network
Smooth	Low res. roads, rivers	High res. roads	Road networks
Polygonal	Man-made structures	City streets	Street Networks
Planar	Planar structures	City streets	Street Networks
Rectilinear	Roof tops, parking lots	City streets	Buildings

In collaboration with ETH-Zürich, we also have developed a new approach to snake-based delineation. This method allows a user to outline an open contour by specifying only very distant end points and allows the system to propagate edge information from the extremities toward the center. This substantial improvement over traditional snakes is documented in Appendix B.

We have evaluated the effectiveness of our package by instrumenting the code to record the amount of user intervention. We have found that using our snakes to model high-resolution roads leads to a five-fold reduction in effort as measured by the number of mouse clicks or mouse travel time. These results are documented in Appendix C.

Furthermore, we have developed a constrained-optimization scheme that allows us to impose hard constraints on our snakes. For example, we can ensure that two snakes are at a given distance from each other or that the altitude along the model of a river decreases monotonically [7].

4 CONTEXT-BASED ARCHITECTURE

Thirty years of IU research have produced an enormous number of computer vision algorithms, many of which have demonstrated reliable performance at solving particular tasks in restricted

domains. However, the development of computer vision systems that are reliable in more general circumstances has proved elusive [8]. It is in response to this situation that a framework is offered within which computer vision algorithms of specialized competence can be used and integrated with other such algorithms to produce a reliable vision system that operates effectively in a broader context than any of its individual component algorithms. Recently, other authors have stressed the use of context to aid image interpretation [9, 10, 11].

The site model construction and editing systems being developed within the RADIUS program comprise numerous computer vision algorithms, each tailored to accomplish a particular task under a particular set of circumstances. The goals of these algorithms may overlap, or the algorithms may even duplicate each other's goals, but the assumptions that they make and the data with which they operate will often differ. In automatically choosing suitable algorithms and parameter settings to solve particular IU tasks, it is important to minimize the understanding of IU technology that is required of a user who endeavors to use the algorithms. Within the RADIUS Program this concern is paramount because of the need to put advanced IU technology in the hands of casual users, specifically the image analysts.

Our strategy for integrating such a collection of computer vision algorithms is based on prior work in context-based vision at SRI [12, 3, 13]. We represent explicitly the assumptions made by each algorithm, and use the context of the present task to select the most appropriate algorithms for solving that task. By doing so, we seek to avoid the source of many failures of computer vision techniques — the employment of an algorithm outside the bounds of its intended domain of competence. We have implemented within RCDE a prototype of a context-based architecture (CBA), which offers a design methodology with broad applicability. For example, CBA is an attractive framework for organizing a site model construction system using many algorithms that extract different features under different circumstances.

We have integrated a prototype system into RCDE, known as the Hierarchical Update and Build (HUB) System. We have chosen to use logic programming to implement the context-based architecture because:

- It provides a natural declarative language for expressing the constraints.
- Unification provides a powerful mechanism for matching the contextual constraints (as encoded in the rules) to the current context.
- The logical backchaining of a rule based system provides the ability to search the rule base for algorithms that are applicable.

It is clear that the performance of an IU system employing the context-based architecture could also be attained by integrating the same computer vision algorithms via more traditional methods. However, the explicit representation of contextual constraints affords the following additional benefits that would be lost to a purely functional integration.

Task specification: The context-based architecture allows the user to specify the task to be accomplished, leaving the selection of specific algorithms to be decided by the system. For example, the user can state that he would like the system to construct a 3–D model of a building, and the system would decide which of several building extraction algorithms would be most appropriate given the currently available imagery and auxiliary data. The user can make effective use of the IU system while possessing little knowledge of the capabilities and limitations of the individual computer vision algorithms.

Choosing parameters: The context rules are used to establish algorithm parameter settings in the same way that they delimit the range of applicability. While computer vision algorithms can often compute their parameter settings from data at runtime, the context rules provide a uniform means for all algorithms to specify how their parameters are to be determined.

Incremental integration: When building large systems in an evolutionary fashion, it can be difficult to add new capabilities without jeopardizing the integrity of existing capabilities. The modular decomposition of the context rules allows the developer to integrate a new algorithm by adding a packet of rules governing its use, without modifying existing code.

Choosing imagery: It is sometimes important to identify the imagery that is most likely to allow an algorithm to yield a desired result, rather than to choose an algorithm to run on a preselected image. The context rules already encode the information necessary to make this determination – they can be used to answer this question by fixing the algorithm and allowing the image to be a variable in the query. In fact, the context-rule base can be used to answer both questions simultaneously, finding the best combination of algorithms and images to satisfy a given task.

The HUB is described in more detail in Appendix D.

To complement the HUB, we have also investigated the issue of parameter learning as a function of image content, collateral data, and task requirements. We have proposed the framework described in Appendix E that lays the foundation for new learning mechanisms to be developed and tested. We have also offered solutions to some of the subproblems that arise: how to define similarity of context vectors in which elements are both numerical and categorical, how to choose

among the multiple parameter vectors that might be retrieved from the data base, and how to update the data base with experience gained through continual use of the feature extraction system. We have implemented and tested an initial design and demonstrated successful performance, within RCDE, using the MBO algorithms.

5 CONCLUSION

The RADIUS Program can benefit greatly from the use of more highly automated means for constructing and updating 3-D site models.

Research on model-based optimization has led to the development of tools that increase the degree of automation and precision that is possible. These tools have been implemented in the RCDE and are being incorporated in the RADIUS Testbed. Further use of these tools within the Testbed will undoubtedly lead to new ideas and additional improvements in the site-model construction process.

Work on the context-based vision paradigm has led to the development of an architecture for integrating independently developed IU algorithms within a single system. The architecture provides the additional benefit of allowing the image analyst to specify a desired result, rather than a specific procedure to be followed, in carrying out a site model construction task.

Together, these developments, which were beyond the state of the art at the outset of this project four years ago, have dramatically increased the feasibility of constructing and maintaining 3-D site models for use within the RADIUS Testbed.

A MODEL-BASED OPTIMIZATION: AN APPROACH TO FAST, ACCURATE, AND CONSISTENT SITE MODELING FROM IMAGERY

Author: P. Fua

To appear in *RADIUS: Image Understanding for Intelligence Imagery*, Oscar Firschein and Thomas M. Strat, editors, Morgan Kaufmann Publishers, 1997.

1 INTRODUCTION

Model-Based Optimization (MBO) is a paradigm in which an objective function is used to express both geometric and photometric constraints on features of interest. A parametric model of a feature (such as a road, a building, or coastline) is extracted from one or more images by automatically adjusting the model's state variables until a minimum value of the objective function is obtained. The optimization procedure yields a description that simultaneously satisfies (or nearly satisfies) all constraints, and, as a result, is likely to be a good model of the feature.

The deformable models we use here are extensions of traditional snakes [4, 5, 6]. They are polygonal curves or facetized surfaces to which is associated an objective function combining an "image term" that measures the fit to the image data and a regularization term that enforces geometric constraints.

Because features and surfaces are all uniformly modeled, we can refine several models simultaneously and enforce geometric and semantic constraints between objects, thus increasing not only the accuracy but also the consistency of the reconstruction. The ability to apply such constraints is essential for the accurate modeling of complex sites in which objects obey known geometric and semantic constraints. In particular, when dealing with multiple objects, it is crucial that the models be both accurate and consistent with each other. For example, individual components of a building can be modeled independently, but to ensure realism, one must guarantee that they touch each other in an architecturally feasible way. Similarly, when modeling a cartographic site from aerial imagery, one must ensure that the roads lie on the terrain—and not above or below it—and that rivers flow downhill. To that end, we have developed a constrained-optimization scheme that allows us to impose hard constraints on our snakes at a very low computational cost while preserving their convergence properties.

We first introduce our generalized snakes. We then present our constrained-optimization scheme. Finally, we demonstrate its ability to enforce geometric constraints upon individual snakes and consistency constraints upon multiple snakes to produce complex and consistent site models.

2 GENERALIZED SNAKES

We model linear features as polygonal curves that may be described either as a sequential list of vertices, or, for more complex objects such as a road network or a 3–D extruded object, described by the network topology. In the latter case, to describe the object completely, one must supply not only the list of vertices but also a list of "edges" that defines the connectivity of those vertices. In addition, with some of these complex objects, one can also define "faces," that is, circular lists of vertices that must be constrained to remain planar.

Similarly, we model the terrain on which these features rest as triangulated surface meshes whose shape is defined by the position of vertices and can be refined by minimizing an objective function.

Our ultimate goal is to accommodate the full taxonomy of those "generalized snakes" described by Table 2. The algorithms described here are implemented within the RADIUS Common Development Environment (RCDE) [14]. The system has been installed at the National Exploitation Laboratory where it is used in a quasi-operational mode by professional image analysts.

Furthermore, we have evaluated the effectiveness of our package by instrumenting the code to record the amount of user intervention. We have found that using our snakes to model high-resolution roads leads to a fivefold reduction in effort as measured by the number of mouse clicks or mouse travel-time. These results are documented in the Appendix.

Table 2: Snake taxonomy. The columns represent different types of snakes and the rows different kinds of constraints that can be brought to bear. The table entries are examples of objects that can be modeled using these combinations.

Constraints/Type	Simple curve	Ribbon curve	Network	Meshes
Smooth	Low res. roads, rivers	High res. roads	Road network	Terrain
Polygonal	Man-made structures	City streets	Street networks	
Planar	Planar structures	City streets	Street networks	
Rectilinear	Roof tops, parking lots	City streets	Buildings	

2.1 Polygonal Snakes

A simple polygonal snake, C, can be modeled as a sequential list of vertices, that is, in two dimensions, a list of 2–D vertices $\mathcal{S}_2$ of the form

$$\mathcal{S}_2 = \{(x_i\, y_i),\ i = 1, \ldots, n\}\ , \tag{1}$$

and, in three dimensions, a list of 3–D vertices $\mathcal{S}_3$ of the form

$$\mathcal{S}_3 = \{(x_i\, y_i\, z_i),\ i = 1,\ldots,n\}\ . \tag{2}$$

In this paper, we refer to S, the vector of all x, y, and z coordinates of the 2–D or 3–D vertices that define the deformable model's shape as the model's *state vector.*

In the 2–D case, the "image energy" of these curves—the term we try to minimize when we perform the optimization is taken to be

$$\mathcal{E}_I(C) = -\frac{1}{|C|}\int_0^{|C|} |\nabla I(\mathbf{f}(s))|\, ds, \tag{3}$$

where I represents the image gray levels, s is the arc length of C, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x,y) in the image, and $|C|$ is the length of C. In practice, $\mathcal{E}_I(C)$ is computed by integrating the gradient values $|\nabla I(\mathbf{f}(s))|$ in precomputed gradient images along the line segments that connect the polygonal vertices.

In the 3–D case, illustrated by Figures 1 and 2(a), $\mathcal{E}_I(C)$ is computed by projecting the curve into a number of images, computing the image energy of each projection, and summing these energies.

2.2 Smooth Snakes and Ribbons

These snakes are used to model smoothly curving features such as roads or ridgelines.

2–D curves. Following Kass *et al.* [5], we choose the vertices of such curves to be roughly equidistant and add to the image energy $\mathcal{E}_I$ a regularization term $\mathcal{E}_D$ of the form

$$\begin{aligned}\mathcal{E}_D(C) &= \mu_1 \textstyle\sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ &+\mu_2 \textstyle\sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2\end{aligned} \tag{4}$$

and define the "total energy" $\mathcal{E}_T$ as

$$\mathcal{E}_T(C) = \mathcal{E}_D(C) + \mathcal{E}_I(C)\ . \tag{5}$$

The first term of $\mathcal{E}_D$ approximates the curve's tension, and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity. To perform the optimization we could use the steepest or conjugate gradient, but it

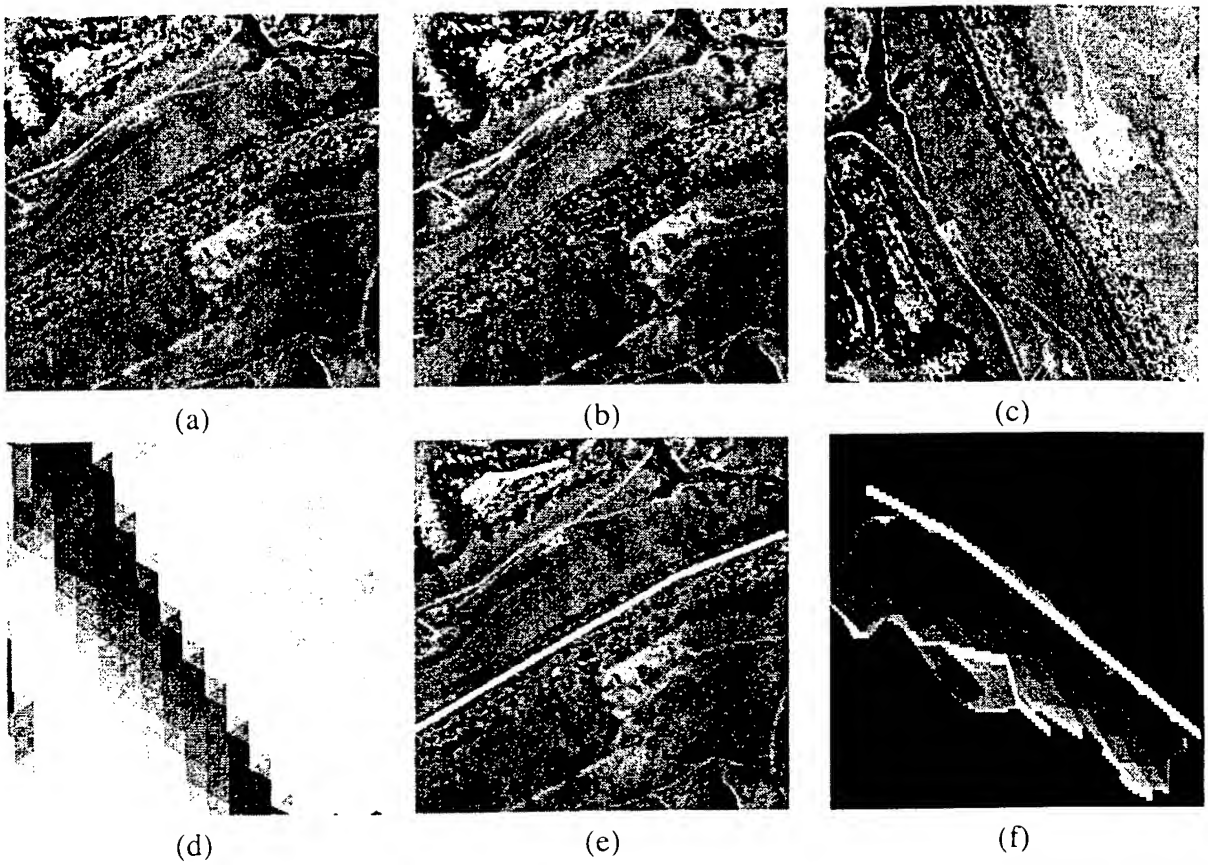


Figure 1: Rugged terrain with sharp ridge lines. (a,b,c) Three images of a mountainous site. (d) Shaded view of an initial terrain estimate. (e) Rough polygonal approximation of the ridgelines overlaid on image (a). (f) The terrain and ridgeline estimates viewed from the side (the scale in z has been exaggerated).

would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0. \\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} + \frac{\partial \mathcal{E}_I}{\partial S}. \end{aligned} \tag{6}$$

where $\mathcal{E}$ is the energy of Equation a-3, α the viscosity of the medium, and S the state vector that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation a-4 is quadratic, its derivative with respect to S is linear, and therefore Equation a-5 can be rewritten as

$$K_S S_t + \alpha(S_t - S_{t-1}) = -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}}$$

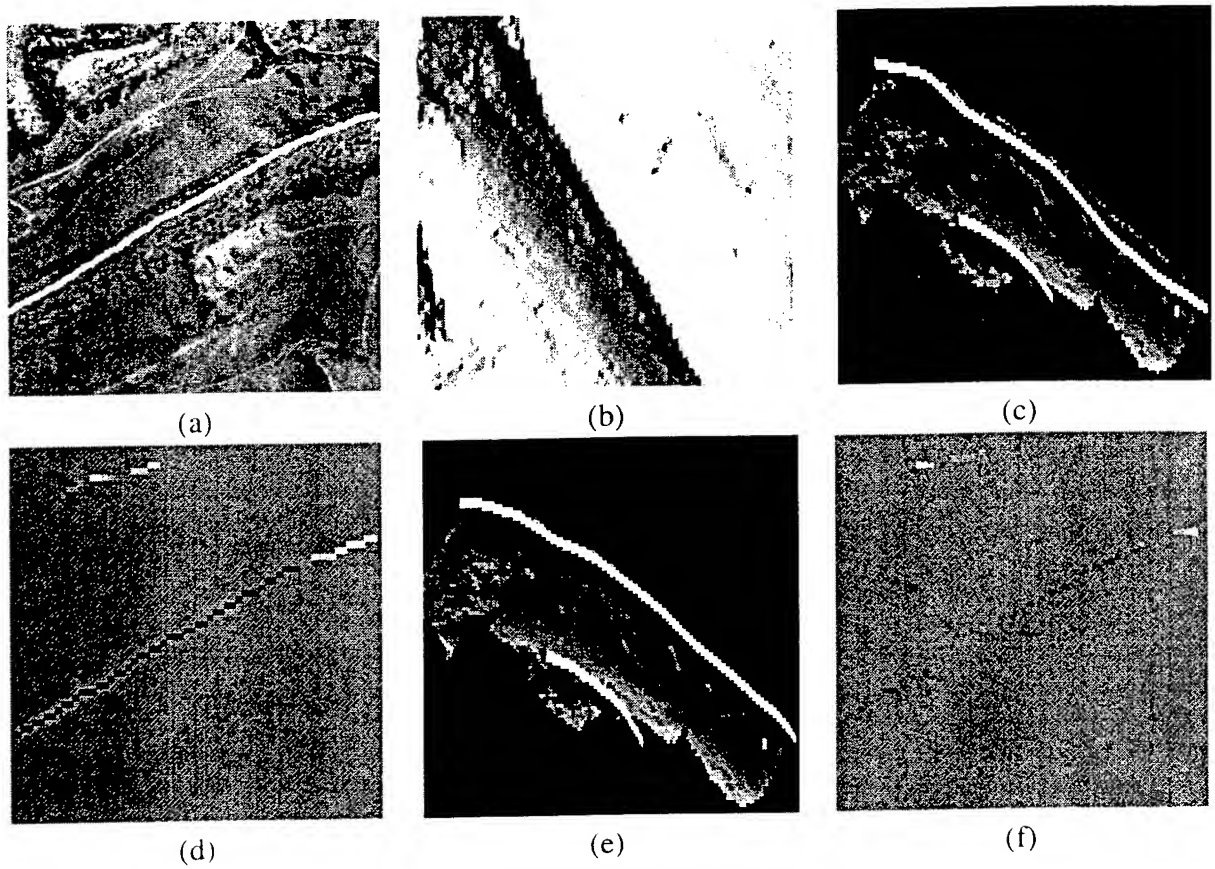


Figure 2: Recovering the 3–D geometry of both terrain and ridges. (a) Refined ridgeline after 3–D optimization. (b) Shaded view of the terrain after refinement. (c) Side view of the ridgeline and terrain after independent optimization of each one. Note that the shape of the ridgeline does not exactly match that of the terrain. (d) Differences of elevation between the recovered ridgeline and the underlying terrain. The image is stretched so that black and white represent errors of minus and plus 80 feet, respectively. (e) Side view after optimization under consistency constraints. (f) Corresponding difference of elevation image stretched in the same fashion as (d).

$$\Rightarrow (K_S + \alpha I)S_t = \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} . \tag{7}$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S.$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation a-6 as a set of two differential equations of the form

$$(K + \alpha I)V - t = \alpha V_{t-1} - \left.\frac{\partial \mathcal{E}_I}{\partial V}\right|_{V_{t-1}} . \tag{8}$$

where V stands for either X or Y, the vectors of the x and y vertex coordinates, and K is a pentadiagonal matrix. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice, α is computed in the following manner. We start with an initial step size Δ_p, expressed in pixels, and use the following formula to compute the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p} \left| \frac{\partial \mathcal{E}}{\partial S} \right|, \tag{9}$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the nonlinear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This procedure is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

3–D curves. To extend the smooth snakes to three dimensions, we add one term in z to the deformation energy of Equation a-4. Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation a-6 as a set of three differential equations of the form of Equation a-7, where V now stands for either X, Y, or Z, the x, y, or z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{E}_I(C)$ is computed by summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S by using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation a-7.

Ribbons 2–D snakes can also be extended to describe ribbon-like objects such as roads in aerial images. A ribbon snake is implemented as a polygonal curve forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The list of vertices can be written as

$$S_2 = \{(x_i\ y_i\ w_i)\},\ i = 1, \ldots, n\} \ . \tag{10}$$

The state vector S becomes the vector of all x, y, and w and the average edge strength the sum of the edge strengths along the two boundary curves. Since the width of roads tends to vary gradually, we add an additional energy term of the form

$$\begin{aligned} \mathcal{E}_W(C) &= \sum_i (w_i - w_{i-1})^2 \qquad (11) \\ \Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} &= LW, \end{aligned}$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}_(C) = \lambda_D \mathcal{E}_D(C) + \lambda_W \mathcal{E}_W(C) + \lambda_G \mathcal{E}_I(C) \ ,$$

where λ_D and λ_W weigh the contributions of the two geometric terms. At each iteration the system must solve the three differential equations in the form of Equation a-7, where V now stands for either X, Y, or W, the x, y, or w vertex coordinates.

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector of all x, y, z, and w and, at each iteration, the system must solve four differential equations, one for each coordinate.

2.3 Network Snakes

The 2–D and 3–D "network snakes" are a direct extension of the polygonal snakes of Section 2.1.

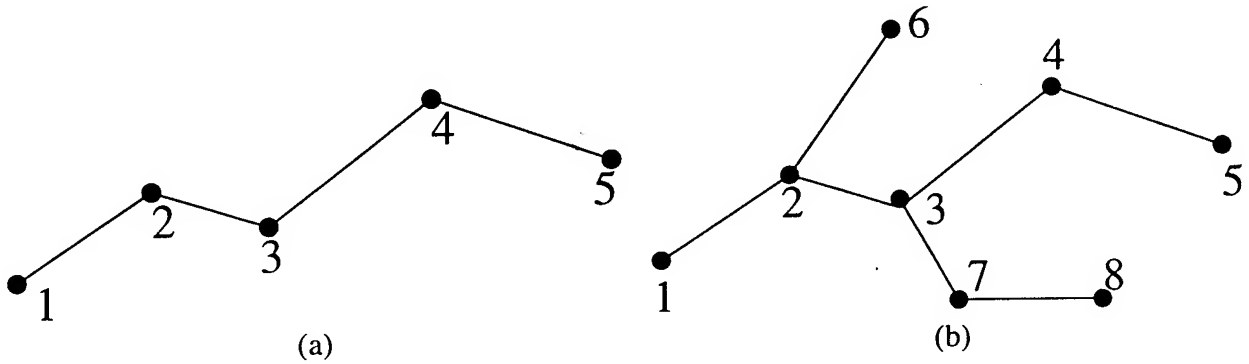


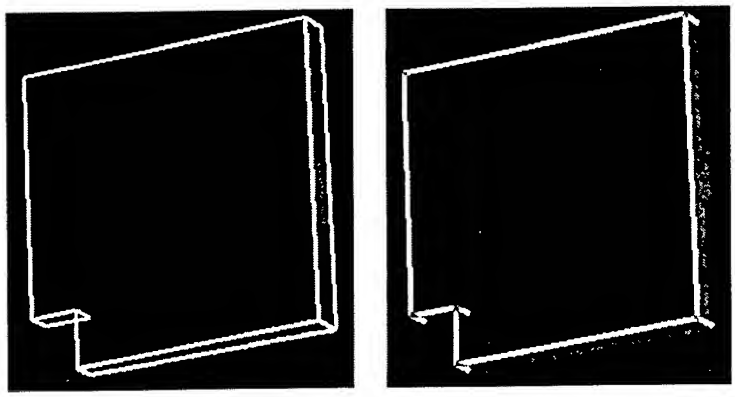
Figure 3: Snake topology. (a) A simple polygonal curve described by a sequential list of vertices v_i, $1 \leq i \leq 5$. (b) A network described by a list of vertices v_i, $1 \leq i \leq 8$, and a list of edges–((1 2) (2 3) (3 4) (4 5) (2 6) (3 7) (7 8)).

In the 2–D case, the extension is straightforward. A network snake is now defined by a list of n vertices $\mathcal{S}$ as before and a list of edges $\mathcal{A} = \{(i, j)$ where $1 \leq i \leq n$ and $1 \leq j \leq n\}$. Figure 3

depicts such a network snake. $\mathcal{E}_I(C)$ is computed as

$$\mathcal{E}_I(C) = \sum_{(i,j)\in\mathcal{A}} \mathcal{E}_I^{i,j} / \sum_{(i,j)\in\mathcal{A}} L^{i,j} \ , \tag{12}$$

where $\mathcal{E}_I^{i,j}$ is the sum of the edge gradients along the $((x_i, y_i)(x_j, y_j))$ segment and $L^{i,j}$ is its length. The snake is optimized using either steepest gradient descent or conjugate gradient.



(a) (b)

Figure 4: Edge visibility. (a) An RCDE "extruded object." Only the visible faces—that is, those whose normal is oriented toward the viewer—are drawn. Note that this heuristic does not account for nonconvexity, and as a result the faces in the lower left corner of the image are improperly drawn. (b) The network snake generated to optimize the extruded object. It includes roof edges and vertical wall edges. The edges at the back of the building are not drawn—and not used during the computations involving these views—because they belong to hidden faces. The edges at the base of the building are treated as invisible because their appearance is unreliable in typical imagery.

In the 3–D case, one must take into account the fact that not all the network's edges are visible in all views. As a result one must also provide, for each projection of the snake into all the images, a list of visible edges. We compute this list by using the face-visibility methods embedded in RCDE as shown in Figure 4.

The number of degrees of freedom of generic 3–D networks can be reduced by forcing them to be planar. We do this either by defining a plane of equation

$$z = ax + by + c \tag{13}$$

and imposing that the vertices lie on such a plane or imposing planar constraints on sets of four vertices using the constrained-optimization approach introduced in Section 3.1. In both cases, we replace the n degrees of freedom necessary to specify the elevation of each vertex by the three

degrees of freedom required to define the plane.

These 3–D networks can be further specialized to handle objects that are of particular interest in urban environments: trihedral corners found on building roofs and extruded objects that are used in RCDE to model building outlines. In Figure 5, we show several buildings modeled by roughly entering their outlines within RCDE and optimizing the shapes in three views simultaneously by using our extruded snakes. The use of the snakes has allowed us to perform this task much faster than we would have if we had to precisely delineate all five buildings by hand. To produce this result, we have used the constrained-optimization technique of Section 3.1 to constrain the "wall" edges to remain vertical. We can also constrain the "roof outline" to be planar and the "roof edges" to form 90-degree angles. These constraints greatly reduce the number of degrees of freedom and allow for better convergence properties.

2.4 3–D Surface Meshes

Given the task of reconstructing a surface from multiple images whose vantage points may be very different, we need a surface representation that can be used to generate images of the surface from arbitrary viewpoints, taking into account self-occlusion, self-shadowing, and other viewpoint-dependent effects. Clearly, a single image-centered representation is inadequate for this purpose. Instead, an object-centered surface representation is required.

Many object-centered surface representations are possible. However, practical issues are important in choosing an appropriate one. First, the representation should be general-purpose in the sense that it should be possible to represent any continuous surface, closed or open, and of arbitrary genus. Second, it should be relatively straightforward to generate an instance of a surface from standard data sets such as depth maps or clouds of points. Finally, there should be a computationally simple correspondence between the parameters specifying the surface and the actual 3-D shape of the surface, so that images of the surface can be easily generated, thereby allowing the integration of information from multiple images.

A regular 3–D triangulation is an example of a surface representation that meets the criteria stated above, and is the one we have chosen for our previous work. In our implementation, all vertices except those on the edges have six neighbors and are initially regularly spaced. Such a mesh defines a surface composed of three-sided planar polygons that we call triangular facets, or simply facets. Triangular facets are particularly easy to manipulate for image and shadow generation; consequently, they are the basis for many 3-D graphics systems. These facets tend to form hexagons and can be used to construct virtually arbitrary surfaces. Finally, standard triangulation

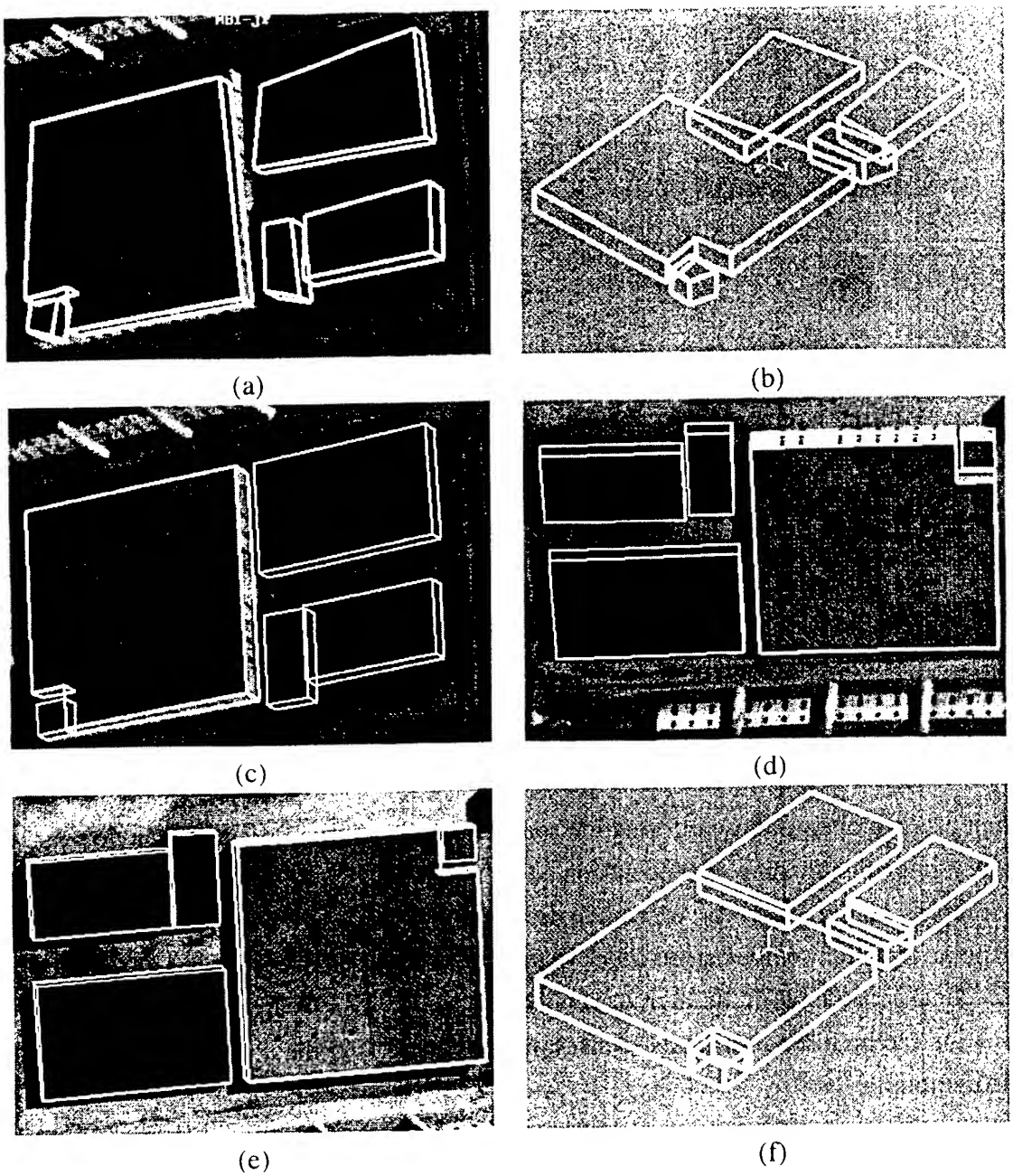
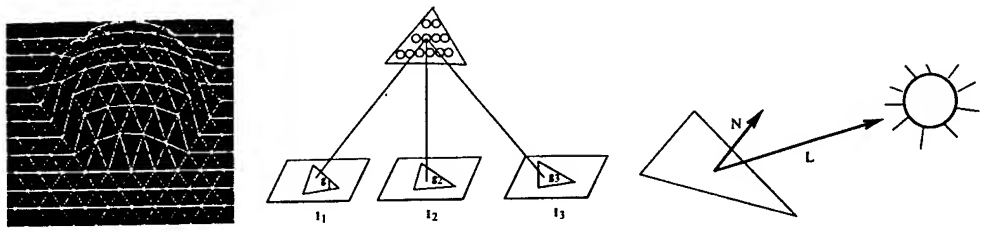


Figure 5: Buildings modeled by entering rough models within RCDE and optimizing them using the extruded snakes. (a) Rough initial sketches overlaid on one of the images. (b) A view from a different perspective. (c,d,e) Final building outlines overlaid on the three images we used to perform the 3–D optimization. (f) A view of the buildings from the perspective of (b).

algorithms can be used to generate such a surface from noisy real data [15, 16].



(a) (b) (c)

Figure 6: Mesh representation and computation of the image terms of the objective function: (a) Wireframe representation of the mesh. (b) Facets are sampled at regular intervals; the circles represent the sample points. The stereo component of the objective function is computed by summing the variance of the gray level of the projections of these sample points, the g_is. (c) Each facet's albedo is estimated using its normal N, the light source direction L, and the average gray level of the projection of the facet into the images. The shading component of the objective function is the sum of the squared differences in estimated albedo across neighboring facets.

Sources of information. A number of information sources are available for the reconstruction of a surface and its material properties. Here, we consider two classes of information.

The first class comprises those information sources that do not require more than one image, such as texture gradients, shading, and occlusion edges. When using multiple images and a full 3-D surface representation, however, we can do certain things that cannot be done with a single image. First, the information source can be checked for consistency across all images, taking occlusions into account. Second, when the source is consistent and occlusions are taken into account, the information can be fused over all the images, thereby increasing the accuracy of the reconstruction.

The second class comprises those information sources that require at least two images, such as the triangulation of corresponding points between input images (given camera models and their relative positions). Generally speaking, this source is most useful when corresponding points can be easily identified and their image positions accurately measured. The ease and accuracy of this correspondence can vary significantly from place to place in the image set, and depend critically on the type of feature used. Consequently, whatever the type of feature used, one must be able to identify where in the images that feature provides reliable correspondences, and what accuracy one can expect.

The image feature that we have chosen for correspondence (although it is by no means the only one possible) is simply intensity in radiometrically corrected images—for example, by filtering

them. Clearly, intensity can be a reliable feature only when the albedo varies quickly enough on the surface and, consequently, the images are sufficiently textured.

Simple correlation-based stereo methods often use fixed-size windows in images to measure disparities, which will in general yield correct results only when the surface is parallel to the image plane. Instead, we compare the intensities as projected onto the facets of the surface. Consequently, the reconstruction can be significantly more accurate for slanted surfaces. Some correlation-based algorithms achieve similar results by using variable-shaped windows in the images [17, 18, 19, 20, 21]. However, they typically use only image-centered representations of the surface.

Our approach is much more closely related to the least-squares approaches advocated by Wrobel [22] and Heipke [23], who both use a 2-1/2–D representation of the surface.

As for the monocular information source, we have chosen to use shading, where shading is the change in image intensity due to the orientation of the surface relative to a light source. We use this method because shading is most reliable when the albedo varies slowly across the surface; this is the natural complement to intensity correspondence, which requires quickly varying albedo. The complementary nature of these two sources allows us to accurately recover the surface geometry and material properties for a wide variety of images.

In contrast to our approach, traditional uses of shading information assume that the albedo is constant across the entire surface, which is a major limitation when applied to real images. We overcome this limitation by improving upon a method to deal with discontinuities in albedo alluded to in the summary of Leclerc and Bobick [24]. We compute the albedo at each facet by using the normal to the facet, a light-source direction, and the average of the intensities projected onto the facet from all images. We use the local variation of this computed albedo across the surface as a measure of the correctness of the surface reconstruction. To see why albedo variation is a reasonable measure of correctness, consider the case when the albedo of the real surface is constant. When the geometry of the mesh is correct, the computed albedo should be approximately the same as the real albedo, and hence should be approximately constant across the mesh. Thus, when the geometry is incorrect, this will generally give rise to variations in the computed albedo that we can take advantage of. Furthermore, by using a *local* variation in the computed albedo, we can deal with surfaces whose albedo is not constant, but instead varies slowly over the surface.

Implementation. The triangulated 3–D mesh of vertices that represents a surface, S, is a hexagonally connected set of vertices such as the one shown in Figure 6(a). The position of a

vertex v_j is specified by its Cartesian coordinates (x_j, y_j, z_j). The mesh can be deformed by varying these coordinates to minimize an objective function that includes terms derived from stereo and shading information. Its state vector S is the vector of all x,y, and z coordinates.

The stereo component of the objective function is derived by comparing the gray levels of the points in all the images for which the projection of a given point on the surface is visible. It is similar to the term proposed by Wrobel [22]. As shown in Figure 6(b), this comparison is done for a uniform sampling of the surface. This method allows us to deal with arbitrarily slanted regions and to discount occluded areas of the surface.

The shading component of the objective function is computed by using a method that does not invoke the traditional constant albedo assumption. Instead, it attempts to minimize the variation in albedo across the surface, and can therefore deal with surfaces whose albedo varies slowly. This term is depicted by Figure 6(c).

The stereo term is most useful when the surfaces are highly textured. Conversely, the shading term is most reliable where the surfaces have little or no texture. To account for this phenomenon, we take the complete objective function, $\mathcal{E}(S)$, to be a weighted average of these two components where the weighting is a function of texture within the projections of individual facets.

In general, $\mathcal{E}(S)$ is a highly nonconvex function of the vertex positions. To minimize $\mathcal{E}(S)$, we use the "snake-type" [5] optimization technique of Section 2.2. We define the total energy of the mesh, $\mathcal{E}_T(S)$, as

$$\mathcal{E}_T(S) = \mathcal{E}_D(S) + \mathcal{E}(S) \tag{14}$$

where $\mathcal{E}_D(S)$ is a regularization term analogous to the one of Equation a-3. In practice, we take $\mathcal{E}_D$ to be a measure of the curvature or local deviation from a plane at every vertex. Because the mesh is regular, $\mathcal{E}_D$ can be approximated by using finite differences as a quadratic form [25]

$$\mathcal{E}_D(S) = 1/2(X^T K X + Y^T K Y + Z^T K Z)\,, \tag{15}$$

where X,Y, and Z are the vectors of the x,y, and z coordinates of the vertices, and K is a sparse and banded matrix. This regularization term serves a dual purpose. First, as before, it "convexifies" the energy landscape when λ_D is large and improves the convergence properties of the optimization procedure. Second, in the presence of noise, some amount of smoothing is required to prevent the mesh from overfitting the data, and wrinkling the surface excessively.

To speed the computation and prevent the mesh from becoming stuck in undesirable local minima, we typically use several levels of mesh sizes—three in the example of Figure 2(b)—to

perform the computation. We start with a relatively coarse mesh that we optimize. We then refine it by splitting every facet into four smaller ones and reoptimizing. Finally, we repeat the split and optimization processes one more time.

3 ENFORCING CONSISTENCY

We now turn to the enforcing of geometric and consistency constraints on the multiple objects that may compose a complex site.

A traditional way to enforce such constraints is to add a penalty term to the model's energy function for each constraint. While this may be effective for simple constraints, this approach rapidly becomes intractable as the number of constraints grows, for two reasons. First, it is well known that minimizing an objective function that includes such penalty terms constitutes an ill-behaved optimization problem with poor convergence properties [26, 27]: the optimizer is likely to minimize the constraint terms while ignoring the remaining terms of the objective function. Second, if one tries to enforce several constraints of different natures, the penalty terms are unlikely to be commensurate and one has to face the difficult problem of adequately weighing the various constraints.

Using standard constrained optimization techniques is one way of solving these two problems. However, while there are many such techniques, most involve solving large linear systems of equations and few are tailored to preserving the convergence properties of the snake-like approaches of Sections 2.2 and 2.4. Exceptions are the approach proposed by Metaxas and Terzopoulos [28] to enforce holonomic constraints by modeling the second-order dynamics of the system and the technique proposed by Amini *et al.* [29] using dynamic programming.

Here, we propose a new approach to enforcing hard constraints on our snakes without undue computational burden while retaining their desirable convergence properties.

3.1 Constrained Optimization in Orthogonal Subspaces

Formally, the constrained optimization problem can be described as follows. Given a function f of n variables $S = \{s_1, s_2, .., s_n\}$, we want to minimize it under a set of m constraints $C(S) = \{c_1, c_2, .., c_m\} = 0$. That is,

$$\text{minimize } f(S) \text{ subject to } C(S) = 0 \ . \tag{16}$$

While there are many powerful methods for nonlinear constrained minimization [27], we know of none that are particularly well adapted to snake-like optimization: they do not take advantage of the locality of interactions that is characteristic of snakes. We have therefore developed a robust two-step approach [30, 7] that is closely related to gradient projection methods first proposed by Rosen [31] and can be extended to snake optimization.

Solving a constrained optimization problem involves satisfying the constraints and minimizing the objective function. For our application, it has proved effective to decouple the two and decompose each iteration into two steps:

1. Enforce the constraints by projecting the current state onto the constraint surface. This involves solving a system of nonlinear equations by linearizing them and taking Newton steps.

2. Minimize the objective function by projecting the gradient of the objective function onto the subspace tangent to the constraint surface and searching in the direction of the projection, so that the resulting state does not stray too far away from the constraint surface.

Figure 7 depicts this procedure. Let C and S be the constraint and state vectors of Equation 16 and A be the $n \times m$ Jacobian matrix of the constraints. The two steps are implemented as follows:

1. To project S, we compute dS such that $C(S+dS) \approx C(S) + A^t dS = 0$ and increment S by dS. The shortest possible dS is found by writing dS as AdV and solving the equation $A^t AdV = -C(S)$.

2. To compute the optimization direction, we first solve the linear system $A^T(S)A(S)\lambda = A^T(S)\nabla f$ and take the direction to be $\nabla f - A\lambda$. This amounts to estimating Lagrange multipliers, that is, the coefficients that can be used to describe ∇f as closely as possible as a linear combination of constraint normals.

These two steps operate in two locally orthogonal subspaces, in the column space of A and in its orthogonal complement, the null space of A^T. Note that $A^T(S)A(S)$ is an $m \times m$ matrix and is therefore small when there are more variables than constraints, which is always the case in our application.

This technique has been used to enforce the geometric constraints in the example of Figure 5. Furthermore, it can be generalized to handle inequality constraints by introducing an "active set strategy." The inequality constraints that are strictly satisfied are deactivated, while those that are

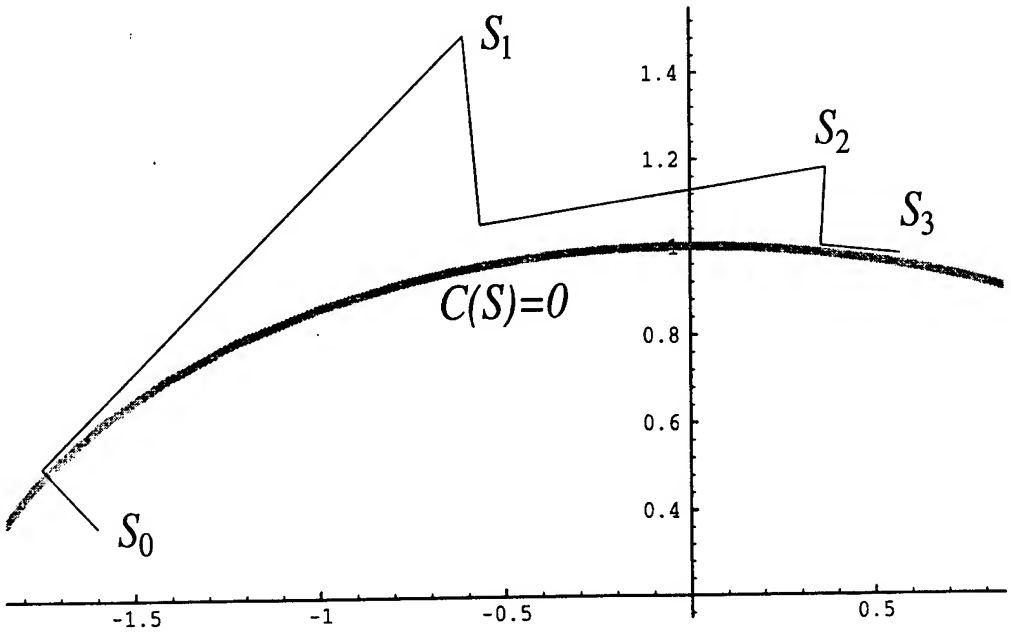


Figure 7: Constrained optimization. Minimizing $(x-0.5)^2+(y-0.2)^2$ under the constraint that $(x/2)^2+y^2=1$. The set of all states that satisfy the constraint $C(S)=0$, i.e. the constraint surface, is shown as a thick gray line. Each iteration consists of two steps: orthognal projection onto the constraint surface followed by a line search in a direction tangent to the surface. Because we perform only one Newton step at each iteration, the constraint is fully enforced after only a few iterations.

violated are activated and treated as equality constraints. This requires additional bookkeeping but does not appear to noticeably slow down the convergence of our constrained-optimization algorithm.

3.2 Constraining Snake Optimization

We could trivially extend the technique of Section 3.1 to the refinement of smooth curves and surfaces by taking the objective function f to be the total energy $\mathcal{E}_T$ of Equation a-3. However, this would be equivalent to optimizing an unconstrained snake by using gradient descent as opposed to performing the implicit Euler steps that so effectively propagate smoothness constraints.

In practice, propagating the smoothness constraints is key to forcing convergence toward desirable answers. When a portion of the snake deforms to satisfy a hard constraint, enforcing regularity guarantees that the remainder of the snake also deforms to preserve it and that unwanted discontinuities are not generated. This is especially true in our application because

many of the constraints we use can be satisfied by moving a small number of vertices, thereby potentially creating "kinks" in the curve or surface that subsequent optimization steps may not be able to remove without getting stuck in local minima.

Therefore, for the purpose of optimizing constrained smooth snakes, we decompose the second step of the optimization procedure of Section 3.1 into two steps. We first solve the unconstrained Dynamics Equation (Equation a-6) as we do for unconstrained snakes. We then calculate the component of the snake step vector—the difference between the snake's current state and its previous one—that is perpendicular to the constraint surface and subtract it from the state vector. The first step regularizes, while the second prevents the snake from moving too far away from the constraint surface.

As in the case of unconstrained snakes, α, the viscosity term of Equation a-6, is computed automatically at the start of the optimization and progressively increased as needed to ensure a monotonic decrease of the snake's energy and ultimate convergence of the algorithm.

Let S be the snake's state vector as described in Sections 2.2 and 2.4. An iteration of the optimization procedure involves the following three steps:

1. Take a Newton step to project S_{t-1}, the current state vector, onto the constraint surface.

$$S_{t-1} \leftarrow S_{t-1} + AdV \text{ where } A^T AdV = -C(S_{t-1}) \ .$$

   If the snake's total energy has increased, back up and increase viscosity.

2. Take a normal snake step by solving

$$(K_S + \alpha I)S_t \ = \ \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \ .$$

3. Ensure that dS, the snake step from S_{t-1} to S_t, is in the subspace tangent to the constraint surface.

$$S_t \leftarrow S_t - A\lambda \text{ where } A^t A\lambda = A^T(S_t - S_{t-1}) \ ,$$

   so that the snake step dS becomes

$$\begin{aligned} dS &= (S_t - A\lambda) - S_{t-1} \\ \Rightarrow A^T dS &= 0. \end{aligned}$$

3.3 Multiple Snakes

Our technique can be further generalized to the simultaneous optimization of several snakes under a set of constraints that bind them. We concatenate the state vectors of the snakes into a composite state vector S and compute for each snake the viscosity coefficient that would yield steps of the appropriate magnitude if each snake was optimized individually. The optimization steps become

1. Project S onto the constraint surface as before and compute the energy of each individual snake. For all snakes whose energy has increased, revert to the previous position and increase the viscosity.
2. Take a normal snake step for each snake individually.
3. Project the global step into the subspace tangent to the constraint surface.

Because the snake steps are taken individually, we never have to solve the potentially very large linear system involving all the state variables of the composite snake but only the smaller individual linear systems. Furthermore, to control the snake's convergence via the progressive viscosity increase, we do not need to sum the individual energy terms. This is especially important when simultaneously optimizing objects of a different nature, such as a surface and a linear feature, whose energies are unlikely to be commensurate so that the sum of these energies would be essentially meaningless.

In effect, the optimization technique proposed here is a decomposition method and such methods are known to work well [27] when their individual components, the individual snake optimizations, are well behaved, which is the case here.

4 CONSISTENT SITE MODELING

We demonstrate the ability of our technique to impose geometric constraints on 2–D and 3–D deformable models using real imagery. More specifically, we address the issue of optimizing the models of 3–D linear features such as roads, ridgelines, rivers, and the terrain on which they lie under the constraint that they be consistent with one another. In Figures 1 and 8 we present two such cases where recovering the terrain and the roads independently of one another leads to inconsistencies.

Because we represent the terrain as a triangulated mesh and the features as 3–D polygonal approximations, consistency can be enforced as follows. For each edge $((x_1, y_1, z_1), (x_2, y_2, z_2))$ of

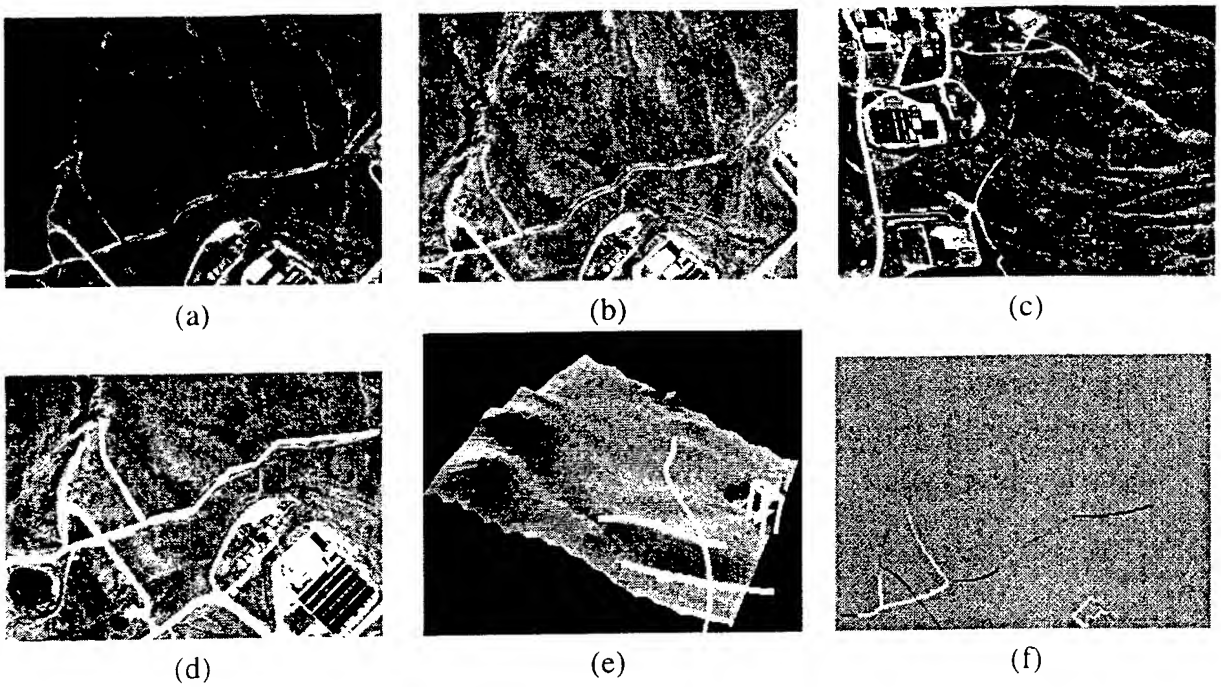


Figure 8: Building a site model. (a,b,c) Three images of a site with roads and buildings. (d) A rough sketch of the road network and of one of the buildings. (e) Shaded view of the terrain with overlaid roads after independent optimization of each. Note that the two roads in the lower right corner appear to be superposed in this projection because their recovered elevations are inaccurate. (f) Differences of elevation between the optimized roads and the underlying terrain. The image is stretched so that black and white represent errors of minus and plus 5 meters, respectively.

the terrain mesh and each segment $((x_3, y_3, z_3), (x_4, y_4, z_4))$ of a linear feature that intersect when projected in the (x, y) plane, the four endpoints must be coplanar so that the segments also intersect in 3–D space. This can expressed as

$$\begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \ . \tag{17}$$

which yields a set of constraints that we refer to as *consistency constraints*.

In Figures 2 and 9, we show that the optimization under the constraints of Equation 17 avoids the discrepancies that result from independent optimization of each feature.

In the example of Figure 2, the "ridge snake" attempts to maximize the average edge gradient along its projections in all three images. In the case of Figures 8 and 9, the roads are lighter than

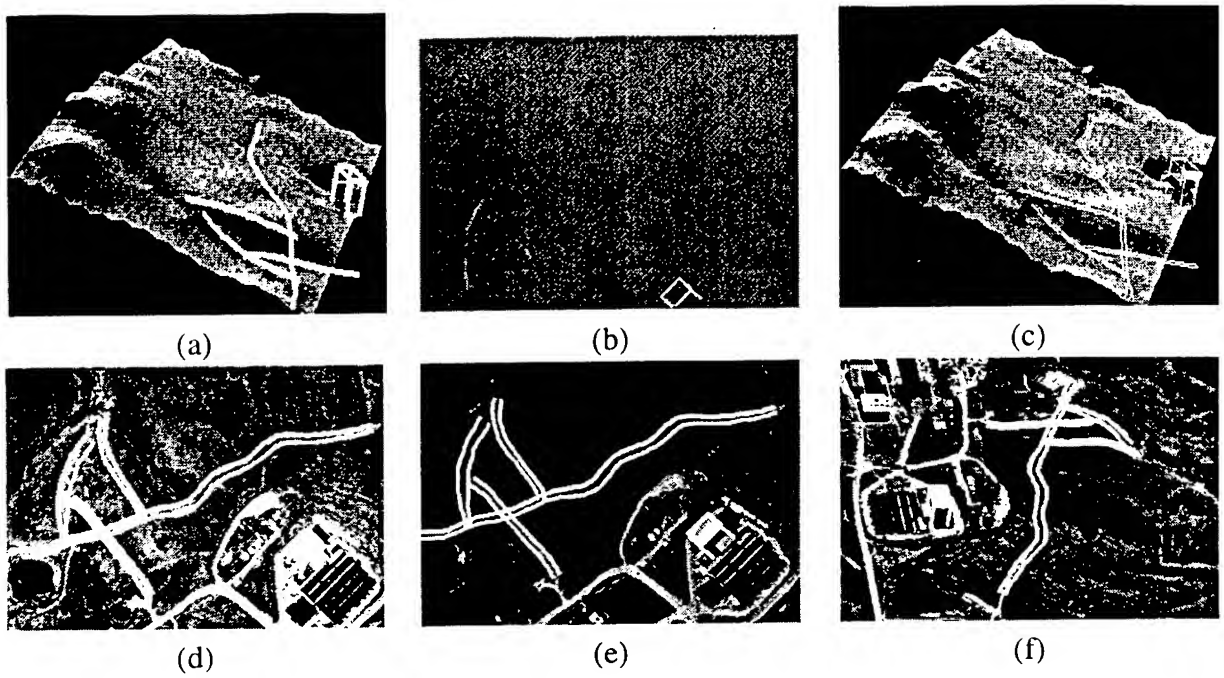
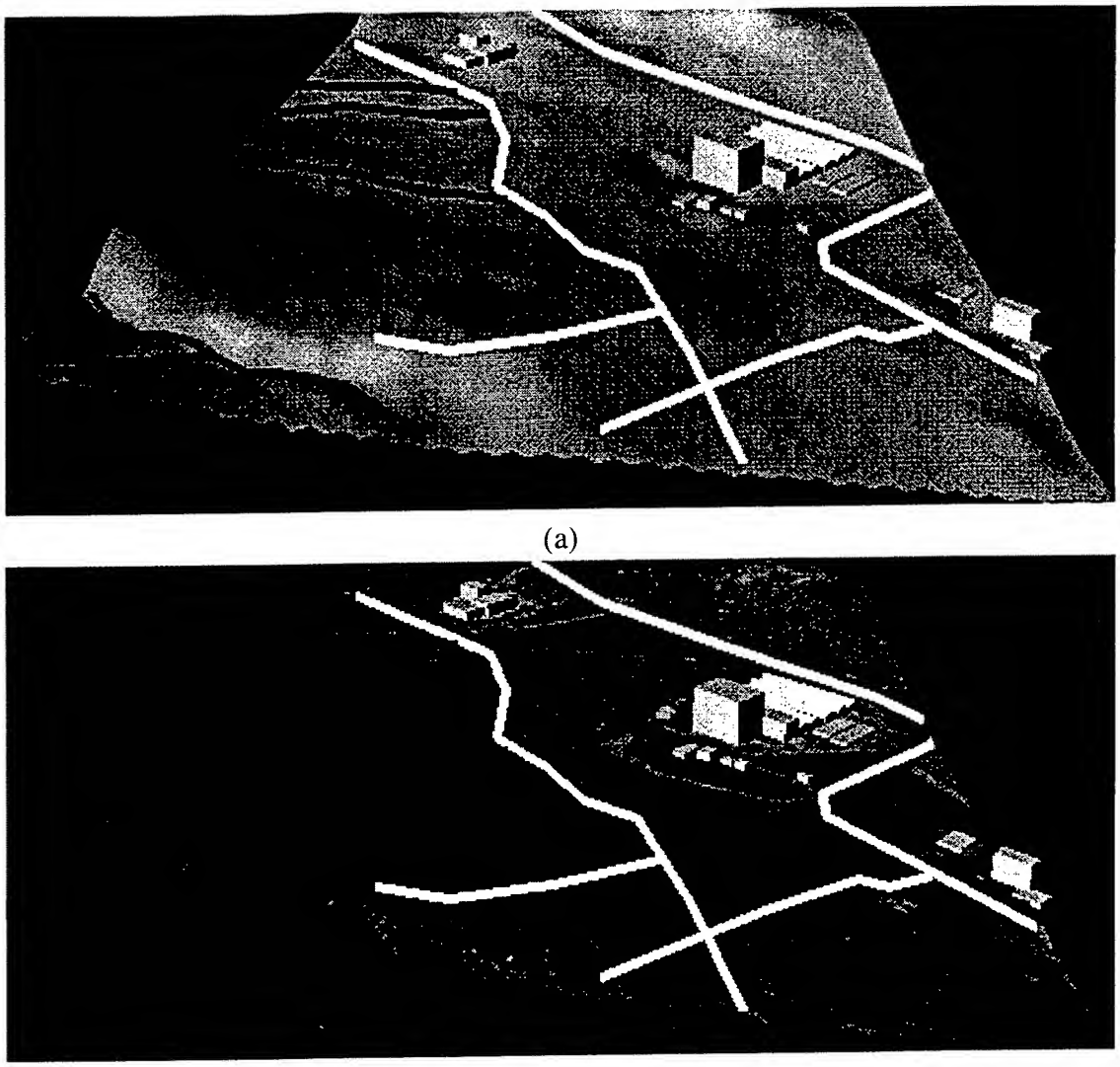


Figure 9: Recovering the 3–D geometry of both terrain and roads. (a) Shaded view of the terrain with overlaid low-resolution roads after optimization under consistency constraints. (b) Corresponding differences of elevation between features and underlying terrain. The image is stretched like the one of Figure 8(f). Note that only the roof of the building is significantly above the terrain. (c) The roads modeled as ribbons overlaid on the terrain. (d,e,f) The optimized roads overlaid on the original images.

the surrounding terrain. At low resolution, they can effectively be modeled as white lines, and the corresponding snakes attempt to maximize image intensity along their projections. At higher resolution, they are better modeled using the 3–D ribbon snakes of Section 2.2. We also introduce a building and use its base to further constrain the terrain. Figures 9(a,b) depict the result of the simultaneous optimization of the terrain and low-resolution roads. By supplying an average width for the roads, we can turn the lines into ribbons and reoptimize terrain and features under the same consistency constraints as before, yielding the result of Figure 9(c).

The case of rivers is somewhat more complex. Like roads, rivers are represented as linear features that must lie on the terrain. But, in addition, the system must ensure that they flow downhill and at the bottom of valleys. By introducing the active set strategy described at the end of Section 3.1, we have been able to impose such constraints and to generate the more complete site model of of Figure 10.

These examples illustrate the ability of our approach to model different kinds of features in a common reference framework and to produce consistent composite models.



(a)

(b)

Figure 10: Composite model. (a) Shaded view of the recovered model. The drainage pattern appears as dark lines, the roads as white lines. (b) Texture mapped view.

5 CONCLUSION

We have presented object modeling techniques for 2–D and 3–D linear features and 3–D surfaces that rely on parametric models and are extensions of traditional snakes. We have shown that we can generate consistent models of complex sites using a constrained optimization method that allows us to enforce hard constraints on these deformable models at a low computational cost.

We believe that this last capability will prove indispensable to automating the generation of complex object databases from imagery, such as the ones required for realistic simulations or intelligence analysis. In such databases, the models must not only be as accurate—that is, true to

the data—as possible but also consistent with each other. Otherwise, the simulation will exhibit "glitches," and the image analyst will have difficulty interpreting the models. Because our approach can handle nonlinear constraints, in future work we will use it to implement more sophisticated constraints than the simple geometric constraints presented here. When modeling natural objects, we intend to take physical laws into account. For example, rivers flow downhill and at the bottom of valleys; these characteristics should be used when modeling both the river and the surrounding terrain. In addition, when modeling man-made objects, we intend to take advantage of knowledge about construction practices, such as the fact that roads do not have arbitrary slopes.

We hope that the technique presented in this paper will eventually form the basis for a suite of tools for modeling complex scenes accurately while ensuring that the model components satisfy geometric and semantic constraints and are consistent with each other.

ACKNOWLEDGMENTS

This article surveys techniques that have been developed in collaboration with Yvan Leclerc and Tom Strat. We wish to thank them for their support, ideas and advice.

A APPENDIX: EVALUATING THE EFFECTIVENESS OF MBO

We have chosen to measure the amount of effort expanded by the human analyst by the number of mouse-clicks and the amount of mouse-travel required to achieve a desired answer. We feel that this is a better measure than, for example, actual computation times because it truly reflects the amount of human interaction and does not depend on the speed of the computer being used.

We first briefly describe the code instrumentation that was required to perform the experiments and then report our results.

a.1 Instrumenting RCDE

We have developed and installed code that captures low-level information from the RCDE user interface about individual actions taken by the analyst. Every mouse-motion associated with making adjustments to object parameters, and every mouse click is captured into an event history. The following information is recorded:

- Object Adjustment Events:
  - object ID
  - event start time
  - adjustment type, for example vertex-xy, vertex-z, or vertex-width
  - 2–D world ID
  - zoom level
  - sequence of time deltas and mouse-position deltas of the form (delta-t dx dy)
- Mouse-Click Events:
  - object ID
  - event start time
  - event ID (for example: zoom-in, zoom-out, recenter, drop-z)
  - 2–D world ID
  - zoom level
  - mouse 2–D world position

This event history is then summarized by a small number of meaningful numbers. Among them are

- number of mouse-clicks
- number of mouse-moves
- total distance mouse moved during adjustments
- total time in adjustment events
- total time in fine adjustments

We have also implemented a metric to estimate the precision of an extracted road by comparing the centerline of the extracted road to the ground truth centerline. For each vertex of the extracted road, the distance to the nearest centerline point of the ground truth centerline is computed. The data is reduced to the following two numbers:

- Mean vertex to ground truth distance
- Maximum vertex to ground truth distance

These seven numbers appear in the figures Section a.2.

a.2 Experimental Results

We used images such as the 7000×7000 one shown in Figure 11 to perform our experiments and chose a set of road segments to be modeled as accurately as possible. We compared four approaches to road delineation:

Hand Hand-tracing using the RCDE interface but neither snakes nor road tracker.

Tracker Using SRI's road tracker [32] to provide the initial sketch in the full-resolution image and then refining it using a ribbon snake [33].

Snake1 Sketching the road using the full resolution image and refining it using a ribbon snake.

Snake2 Sketching the road using a half-resolution version of the image, refining it using a ribbon snake first at half resolution and then at full resolution.

In all cases we used the system's default parameter settings and allowed the user to manually refine the automatically generated results to produce satisfactory delineations. The bar graphs that appear in Figures 12 and 13 are labeled "hand," "tracker," "snake1," "snake2."

We used the hand-traced versions of the roads as our references and the metric discussed above to evaluate the quality of the delineations produced by the three semiautomated approaches. As shown in Figure 12, the results are virtually indistinguishable in terms of average distance, whereas the "tracker" approach does better in terms of maximum distance.

Figure 13 depicts the amount of effort required by each approach, as measured by the number of mouse-clicks and the amount of mouse-travel. The tracker approach appears to be very effective and yields at least a sixfold improvement on all counts except the total number of mouse clicks. This occurs because starting and stopping each operation—automated sketching and snake refinement—require several clicks. This number could be drastically reduced by defining Common-Lisp methods that sequentially perform all these operations with a single mouse-click.

The snake2 approach is almost as effective but requires more effort to provide the initial sketch. This problem could alleviated by using Ziplock Snakes[34] instead of traditional snakes.

For all three semiautomed methods, however, a large portion of the human interaction goes into specifying the width of the road, as the current tools have no way of computing it. Therefore, methods to compute the width of a road given only its centerline would be extremely valuable and should be the object of future research.

In short, by further improving the interface and developing a width-computing algorithms, we should be able to turn the current sixfold reduction of effort into a tenfold to hundredfold one.

Figure 11: An image with two overlaid roads.

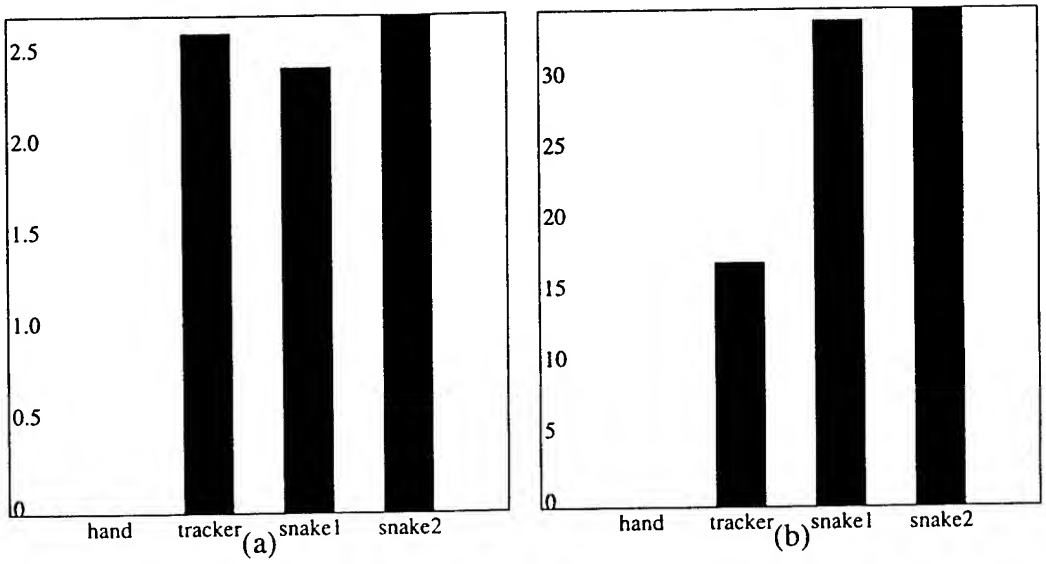


Figure 12: Distance to hand-entered roads. (a) Average distance difference (b) Maximum distance difference. Because the hand-entered results are taken to be the reference, the corresponding distances are zero.

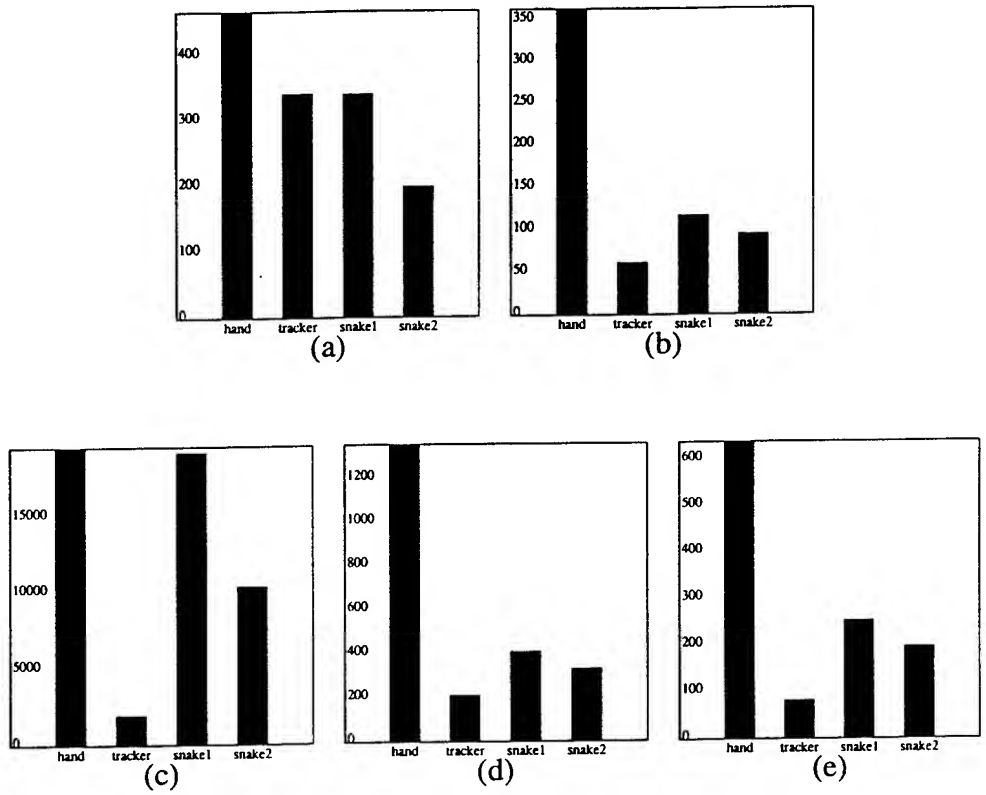
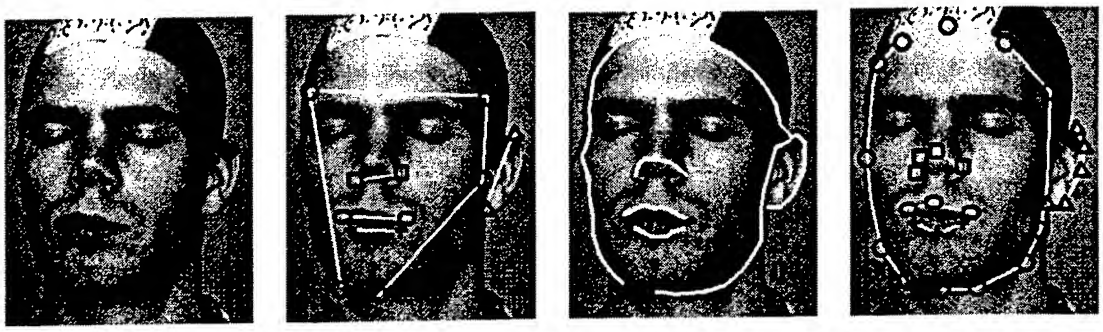


Figure 13: Amount of effort. (a) Number of object clicks. (b) Number of mouse moves. (c) Total mouse distance. (d) Total mouse move time. (e) Total mouse move time for fine adjustments.

B ZIPLOCK SNAKES

Authors: W. Neuenschwander and P. Fua and L. Iverson and G. Székely and O. Kübler
To appear in the International Journal of Computer Vision.

1 INTRODUCTION



(a) (b) (c) (d)

Figure 14: Outlining facial features. (a) A face image with low contrast contours. (b) Five sets of initial points, each denoted by a different symbol. Four of them are pairs of end points while the fifth is a shown as a set of circles. (c) The contours delineated by Ziplock Snakes. (d) The initial delineations that must be supplied to achieve the same result using traditional snakes.

In recent years, snakes have emerged as a powerful tool for semiautomated object delineation. Originated by Terzopoulos, Kass, and Witkin [35, 36] they have since given rise to a large body of literature ([6, 37, 38, 39] among many others) that explores theoretical and implementation issues as well as new applications.

In most of these papers, however, it is assumed that the initial position of the snake is relatively close to the desired solution. While this is a reasonable assumption for applications such as motion tracking [40, 41], it is inappropriate when delineating complex objects from scratch.

Here, we describe a snake approach that allows a user to specify only a few discrete points through which the contour must pass. Our efforts are aimed at simplifying the often repetitive tasks associated with semi-automatic image segmentation. In particular, we intend to eliminate the need to outline the desired structure precisely, that is, to perform a painstaking, almost complete, manual segmentation. As illustrated in Figure 14, considerably fewer control points are needed than for conventional implementations which we refer to as *traditional snakes*.

Traditional snakes are polygonal curves to which is associated an objective function that

combines an "image term" measuring either edge strength or edge proximity, and a regularization term minimizing for tension and curvature. The curve is deformed so as to optimize the objective and, as a result, to match the image edges. The optimization is typically global and takes edge information into account along the whole curve simultaneously. When the snake's initial position is far away from the desired result, the snake often gets stuck in an undesirable local minimum incorporating irrelevant edge information. The minimization problem is solved by treating the snake as a physical system evolving under the influence of a potential that is the sum of an objective function and a dissipation term that enforces convergence. To ensure the convergence of such a local minimization approach, one must supply an initial estimate that is close to the desired answer.

By contrast, in our approach, the optimization progresses from the end points towards the center of the snake, thereby effectively propagating edge information along the curve. The user-supplied end points and the automatically computed edge gradients in their vicinities serve as anchors. They are first used to compute, without using the image potential, an initial state that is approximately correct in each anchor's vicinity. The image term is then "turned on" progressively from the snake's extremities toward its middle section. Still using the end points as anchors, the snake's position is iteratively recomputed until the image term is fully turned on. As a result, the snake is progressively clamped onto an image contour so that it smoothly connects the two end points and has the right orientation at these points. This behavior is analogous to the closing of a ziplock, hence the name of our snakes.

The Ziplock Snakes presented here are similar to the Growing Snakes proposed by [42]. These snakes are initialized with a single point and minimal length. Subsequently the growing snake is slowly extended at the endpoints while the dynamical process iterates. With both Growing and Ziplock snakes, the model information that must be fed into the optimization procedure is reduced to a bare minimum. The model is initially correct at one or two locations respectively, and both methods then try to keep the evolving snake on the object boundary during the optimization. As a result, these snakes avoid getting trapped by local minima and can be used in situations where there are a number of strong nearby features, even though this task is difficult or impossible to perform reliably using Traditional Snakes. However, because we solve a boundary value problem—as opposed to the initial value problem solved by Growing snakes—our optimization scheme need only be controlled by two image-independent parameters that are natural and easy to set, thus making the snake's behavior very predictable.

The *Growing Snake* approach, proposed by [42] is based on an open snake model with free extremities. Since this is the most similar approach to ours we will sketch their algorithm. The

outlining process starts with a short snake which may already lie on a contour. The snake's boundary conditions are set in such a way that both end points are free. At the end points, two new snake segments are attached, hence increasing the snake's length. This new snake is then subject to a fixed number of traditional optimization steps. A post-processing step is then applied to the resulting snake which may cut off those parts of the snake's head and tail which are not aligned with the image contour to be detected. The optimization and post-processing steps are then repeated until both the head and tail are stopped by the post-processing. These steps depend on relatively arbitrary image-based thresholds. To improve the control mechanism for the growing process, Henricsson and Neuenschwander [43] have proposed to combine growing snakes with pre-computed image feature maps derived from energy filters. Here we overcome this difficulty by treating the two extremities of the snake as two growing snakes that stop when they meet.

We first review the properties of traditional snakes and introduce *Ziplock Snakes*. We then compare their respective behaviors, and present results on both synthetic and real images to demonstrate the improved performance of ziplock snakes for initializations that are far away from the desired result. Finally, we propose an extension to ribbon-like structures to facilitate interactive road segmentation.

2 TRADITIONAL SNAKES

The original snakes [36] are modeled as time-dependent 2–D curves defined parametrically as

$$\vec{v}(s,t) = (x(s,t), y(s,t))|_{0 \le s \le 1} ,$$

where s is proportional to the arc length, t the current time, and x and y the curve's image coordinates. The snake deforms itself as time progresses so as to minimize an image potential $E_I(\vec{v})$, with

$$E_I(\vec{v}) = -\int_0^1 P(\vec{v}(s,t))\, ds$$

where $P(\vec{v}(s,t))$ is a function of the image. One typical choice is to take $P(\vec{v}(s,t))$ to be equal to the magnitude of the image gradient, that is

$$P(\vec{v}(s,t)) = |\nabla I(\vec{v}(s,t))| ,$$

where I is either the image itself or the image convolved by a Gaussian kernel.

Whatever the choice of P, $E_I(\vec{v})$ is typically not a convex functional. To overcome this problem, Terzopoulos *et al.* [44] has proposed to introduce a regularization term $E_D(\vec{v})$ that is convex and

to minimize a total energy term $E(\vec{v})$ that is the sum of $E_I(\vec{v})$ and $E_D(\vec{v})$. Using the elastic rod model, $E_D(\vec{v})$ is taken to be

$$E_D(\vec{v}) = \frac{1}{2}\int_0^1 \alpha(s)\left|\frac{\partial\vec{v}(s,t)}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2\vec{v}(s,t)}{\partial s^2}\right|^2 ds\,, \tag{18}$$

where $\alpha(s)$ and $\beta(s)$ are arbitrary functions that regulate the curve's tension and rigidity. In the implementation described below $\alpha(s)$ and $\beta(s)$ are taken to be constant and supplied by the user. We have shown previously [6] that constant α and β can be chosen in a fairly image-independent way.

From variational calculus [45] it is well known that if $\vec{v}(s)$ minimizes $E = E_D + E_I$ and is sufficiently regular, that is at least $C^4(0,1)$, then it must be a solution of the set of two coupled Euler differential equations

$$-\frac{\partial}{\partial s}\left(\alpha(s)\frac{\partial v(s,t)}{\partial s}\right) + \frac{\partial^2}{\partial s^2}\left(\beta(s)\frac{\partial^2 v(s,t)}{\partial s^2}\right) = -\nabla P(x(s,t), y(s,t)) \tag{19}$$

where $v(s,t)$ stands for either $x(s,t)$ or $y(s,t)$. Note that, in order for this equation to have a unique solution [46], one must specify boundary conditions such as the values and derivatives of $v(s,t)$ for $s = 0$ and $s = 1$.

Discretizing Equation (19) using finite differences yields the linear system

$$K \cdot V = F_V \tag{20}$$

where V stands for the vector of either x or y coordinates, K is the stiffness matrix [44] and F_V are the derived image forces. K is not invertible and these equations cannot be solved directly. The Euler differential system of Equation (19) has a unique solution only when boundary conditions are supplied. Without them, it is under-constrained. In Section 3, we show how we can solve this system of equations by using the boundary conditions to turn K into a regular matrix.

Traditional snakes as proposed by [36, 39] for example are only effective when the initial position of $\vec{v}$ is close to the desired solution. However, they are very sensitive to initial conditions. They can easily get caught in local minima when the desired outline presents large concavities that force the snake to extend itself or when there are other edges in the vicinity of the desired one that may "catch" the snake as shown in Figure 15. Furthermore, from a user's point of view, there is not always a simple intuitive way of defining how close an initial state has to be in order to detect the desired object boundary. Figure 16 illustrates this problem.

Figure 15: Sensitivity of traditional snakes to nearby contours. (a) Different initializations: the snake is initialized using a polygon with an increasing number of vertices. (b) The corresponding results: the fact that the two objects are lying close to each other forces the user to outline the desired contour segment precisely. If the snake touches the influence region of a nearby object it will get stuck on the wrong contour.

This is to be expected since the applied traditional optimization algorithm uses a "viscosity" term to stabilize the convergence process along the whole snake. This effectively constrains the result to be the local minimum of the objective function that is in some sense "closest" to the initial position.

3 ZIPLOCK SNAKES

To improve upon traditional snakes' convergence properties, we have developed a boundary-value method with acts like "ziplock" plastic bags. The user specifies end points in the vicinity of a clearly visible edge segment. The system first optimizes the location of the user-supplied points on the gradient image to ensure that they are indeed good edge points, and then calculates the edge directions at these points using both the initial image and its gradient. These anchor elements then become fixed boundary conditions for the dynamical system.

We use the anchor elements to derive an initial position for the snake independently of the image under consideration. This initial state will, in general, be close to the desired answer in the vicinity of the snake end points but nowhere else. We will therefore "turn on" F_V, the image forces of Equation (20), only near the end-points and compute a new position for the snake using the same boundary conditions as before. A longer part of this new solution will be closer to the actual image edge than before; the image forces can thus be turned on in this longer part and the snake's position recomputed. By iterating this process, we eventually turn on the image forces over the whole length of the snake, thereby achieving the propagation of the edge information

from the anchor points to the snake's middle and the progressive ziplock-style clamping. This approach is closely related to perturbation theory [47]: we start with an unperturbed solution of our minimization problem and progressively perturb it by considering more and more of the image forces.

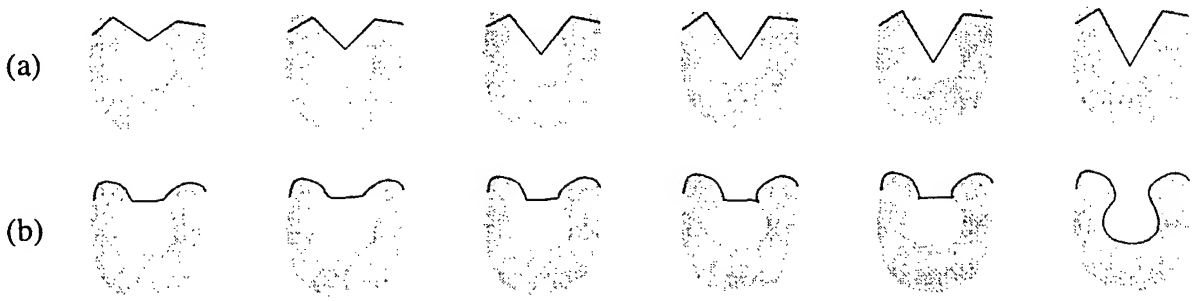


Figure 16: Ill-conditioned behavior of the traditional snakes with respect to initialization. (a) Slightly different initializations: the snake is initialized using a polygon with five vertices. While the first, second, fourth, and fifth vertices are the same for all six situations, the the third vertex moves closer to the shape. (b) The corresponding results: when the third vertex is close enough to the object's border, the snake is able to detect the correct contour (6th image pair). However, it is not intuitively obvious what the threshold is.

In the remainder of this section, we first discuss the use of boundary conditions to solve our minimization problem. We then derive our initial unperturbed snake position from the end points and finally we present the optimization schedule that defines the gradual "turning on" of the image forces.

3.1 Solving the Minimization Problem with Boundary Conditions

As discussed in Section 2, minimizing the snake's energy amounts to solving the Euler differential system of Equation (19) which leads, after discretization, to the semi-linear system of Equation (20). By fixing the curve's end points and specifying the curve's tangent at those points, the system of equations reduces to

$$K^* \cdot V^* = F^*_{V^*}, \tag{21}$$

where V^* stands either for the $n-4$ vector $\mathbf{x}^* = (x_2, \ldots, x_{n-3})$ or $\mathbf{y}^* = (y_2, \ldots, y_{n-3})$, and K^* is an $(n-4) \times (n-4)$ pentadiagonal stiffness matrix that *is* invertible. For additional detail we refer the interested reader to [48]. To our knowledge, other sets of boundary conditions have been implemented by the following authors: e.g. free extremities [36, 49], fixed extremities [50], and periodic boundary conditions for closed snakes [37, 39].

Since $F^*_{V^*}$ depends on the snake's current position, the system is still only semi-linear and cannot, in general, be solved in closed form. Instead, one must use an iteration scheme that can be concisely rewritten as

$$K^* \cdot V^{*[t]} = F^*_{V^*}|_{V^*=V^{*[t-1]}} \tag{22}$$

where t is the iteration index.

3.2 Initialization

To successfully optimize the snake, we must start from an initial position that is approximately correct in the neighborhood of the end points. The easiest way to achieve this result is to solve the homogeneous equations that correspond to the system of Equation (19). By construction, this solution has the specified tangent at the end points and is close to the right answer near these points. As discussed in Section 2, we take α and β to be constant, and the homogeneous system becomes

$$-\alpha\frac{d^2v(s)}{ds^2} + \beta\frac{d^4v(s)}{ds^4} = 0 \tag{23}$$

where v stands for either x or y and $0 \leq s \leq 1$. For both coordinate functions $x(s), y(s)$, the analytical solution of the Equation (23) is of the form

$$v(s) = C_1 + C_2 s + C_3 e^{-\sqrt{\frac{\alpha}{\beta}}s} + C_4 e^{+\sqrt{\frac{\alpha}{\beta}}s} \tag{24}$$

where $C_1, \ldots, C_4$ are integration constants that can be determined from the four boundary conditions $\{v(0)\ ,\ v(1)\ ,\ v'(0)\ ,\ v'(1)\}$. The solution of Equation (23) depends only on the ratio $w = \sqrt{\alpha/\beta}$ and the boundary conditions [35]. The computation displayed above ensures that the initial estimate is already close to the correct solution in the neighborhood of both end points. The same goal could have been achieved by using other interpolation techniques as well, e.g. Bézier-Curves. However, the computation performed here is part of our optimization scheme presented in Section 3.3: Computing Solution (24) of Equation (23) is equivalent to executing a first optimization step. Computing the initial snake position is therefore appealing it is achieved within the same mathematical framework as the rest of the computation.

When interactively defining the snake's end points, which we will subsequently refer to as the head and tail, the user need not be very accurate. Instead, we can ask him to choose the head and tail close to dominant edge fragments. The system then performs a linear search to find the true edge location in the near neighborhood of the selected points. The snake tangent at its head and tail is then computed by optimizing the orientation of a short straight-line snake.

While the tangent direction at the end points can be computed, its orientation may be ambiguous. By default, the boundary conditions are chosen so that the initial snake defines acute angles with the line joining the two end points. Since this heuristic may fail, the interface provides the user with the possibility of flipping the orientation at both ends.

3.3 Optimization Procedure

We start the optimization of the energy term by defining the initial snake as the solution of the homogeneous differential system of Equation (23). At this stage the snake "feels" absolutely no external image forces. Assuming that the user selects both end points near dominant edge fragments in the image, this initialization ensures that the snake already lies close to its optimal position at both ends. During the ongoing iterative optimization process the image potential P is turned on progressively for all the snake vertices, starting from the extremities. We distinguish between *passive* and *active* snake nodes, depending on whether the potential force field F_V^* is turned on for that vertex or not.

As illustrated by Figure 17, we define the *force boundaries* as the furthest active vertices from the end points. These boundaries approach each other during the ongoing optimization process. The

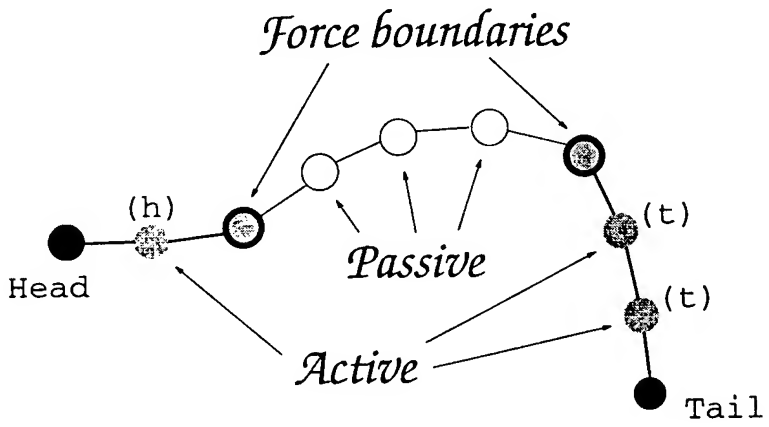


Figure 17: Schematic of the Ziplock Snake during optimization. A Ziplock Snake, fixed at Head and Tail, consists of two parts the *active* and the *passive* vertices. These areas are separated by moving *force boundaries*. The *active* part of the snake is divided up into two segments marked as `(h)` and `(t)` respectively.

simplest way to optimize the snake would be to gradually move the force boundaries from the snake's head and tail towards its center and solve Equation (22) for each new position. However, because Equation (22) is only semi-linear, it cannot be solved in a single step.

Instead we use the following procedure. Each time, the force boundaries are moved, we recompute the shape of the snake's *passive* part by solving the homogeneous Euler Equation 23, using the force boundaries and the *active vertices* on each side to supply the boundary conditions. We introduce a viscosity term $\gamma(s,t)$—similar to the one used by traditional snakes [36]—for the *active* vertices and iteratively solve the modified Equation (22):

$$(K^* + \gamma^{[t]}\,\mathbf{I}) \cdot V^{*[t]} = \gamma^{[t]}\,V^{*[t-1]} + \mathbf{1}\Big|_{\gamma^{[t]}} \cdot F^*_{V^{*[t-1]}} \tag{25}$$

where V^* stands for either $\mathbf{x}^*$ or

$\mathbf{y}^*$ and

$$F^*_{V^{*[t-1]}} = \begin{cases} -\dfrac{\partial P}{\partial x}\Big|_{\left(\mathbf{x}^{*[t-1]},\,\mathbf{y}^{*[t-1]}\right)} & \text{if } V = \mathbf{x} \\ -\dfrac{\partial P}{\partial y}\Big|_{\left(\mathbf{x}^{*[t-1]},\,\mathbf{y}^{*[t-1]}\right)} & \text{if } V = \mathbf{y} \end{cases}$$

The superscript $^{[t]}$ denotes the iteration step and $\mathbf{1}$ is an indicator function for the vector $\gamma^{[t]}$

$$\mathbf{1}\Big|_{\gamma^{[t]}} \in I\!R^{(n-4)\times(n-4)} \quad \text{with} \quad \begin{array}{l} \mathbf{1}_{ij} = 0 \text{ for } i \neq j \\ \mathbf{1}_{ii} = \begin{cases} 1 \text{ if } \gamma_i^{[t]} > 0 \\ 0 \text{ otherwise} \end{cases} \end{array} \tag{26}$$

which has the form of a diagonal matrix.

The viscosity $\gamma(s,t)$ is initialized as $\gamma(s,0) \equiv 0\,, s \in [0,1]$ and recomputed each time the force boundaries move so that the initial displacement of each vertex is on the average of a given a-priori magnitude, typically 1 pixel (see [6]). For $t = 0$, Equation (25 yields the discrete version of the differential Equation (23) with boundary conditions. Hence the computation of the initial estimate $(\mathbf{x}^{*[0]}, \mathbf{y}^{*[0]})^T$ is the first optimization step and fits therefore perfectly into the proposed "ziplock" framework.

Each force boundary is moved individually but at most one vertex per iteration. A force boundary is advanced when we can verify that the motion of the corresponding active section has stabilized. We do this by testing at each iteration step whether the boundary vertex has entered a convergence region well-described by the exponential model $z^{[i]} = \hat{z} + a * e^{-bi}$. Taking three measured values $z^{[0]}, z^{[1]}, z^{[2]}$ the three model parameters $\hat{z}, a, b$ are uniquely determined. Any three non-collinear

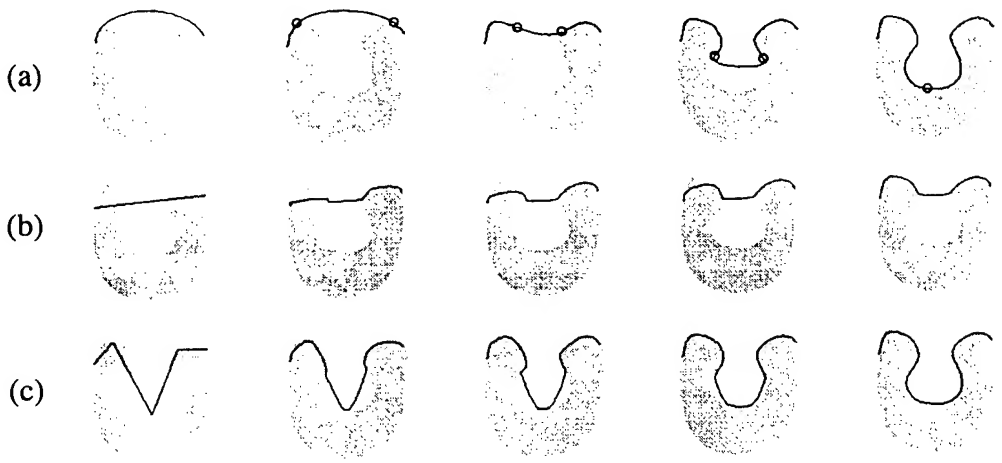


Figure 18: The effect of initialization for ziplock and traditional snakes. The rows show the evolution for different snakes from the initial state in the first column. (a) Evolution of a ziplock snake. The circles indicate the movement of the *force boundaries* during optimization. (b) Traditional snake with same initialization. (c) Traditional snake initialized in three additional points.

coordinates will fit this model giving the convergence point

$$\hat{z} = \frac{z^{[0]}z^{[2]} - z^{[1]}z^{[1]}}{z^{[0]} - 2z^{[1]} + z^{[2]}} \tag{27}$$

To verify convergence, we examine the five most recent positions of the boundary vertex V_k^*. We fit three time samples to the model yielding the according convergence estimates as shown in the table below.

3 time samples	$V_k^{*[t]}, V_k^{*[t-1]}, V_k^{*[t-2]}$	$V_k^{*[t-1]}, V_k^{*[t-2]}, V_k^{*[t-3]}$	$V_k^{*[t-2]}, V_k^{*[t-3]}, V_k^{*[t-4]}$
convergence estimates	$\hat{V}_{k,1}^{*[t]}$	$\hat{V}_{k,2}^{*[t]}$	$\hat{V}_{k,3}^{*[t]}$

If these three estimates fall within a circle of radius smaller than $1/4$ of a pixel then we take this to indicate convergence and advance the corresponding force boundary.

Once both force boundaries have met and all vertices are active, we double the viscosity term γ for each vertex at every 10th iteration. As soon as the average motion of the whole snake falls below $1/10$ pixel the optimization process is stopped.

Figure 19: Evolution of a ziplock snake on the synthetic image of Figure 15. The circles denote the farthest vertices away from the end points for which the image forces are turned on, that is the *force boundaries* of Section 3.3.

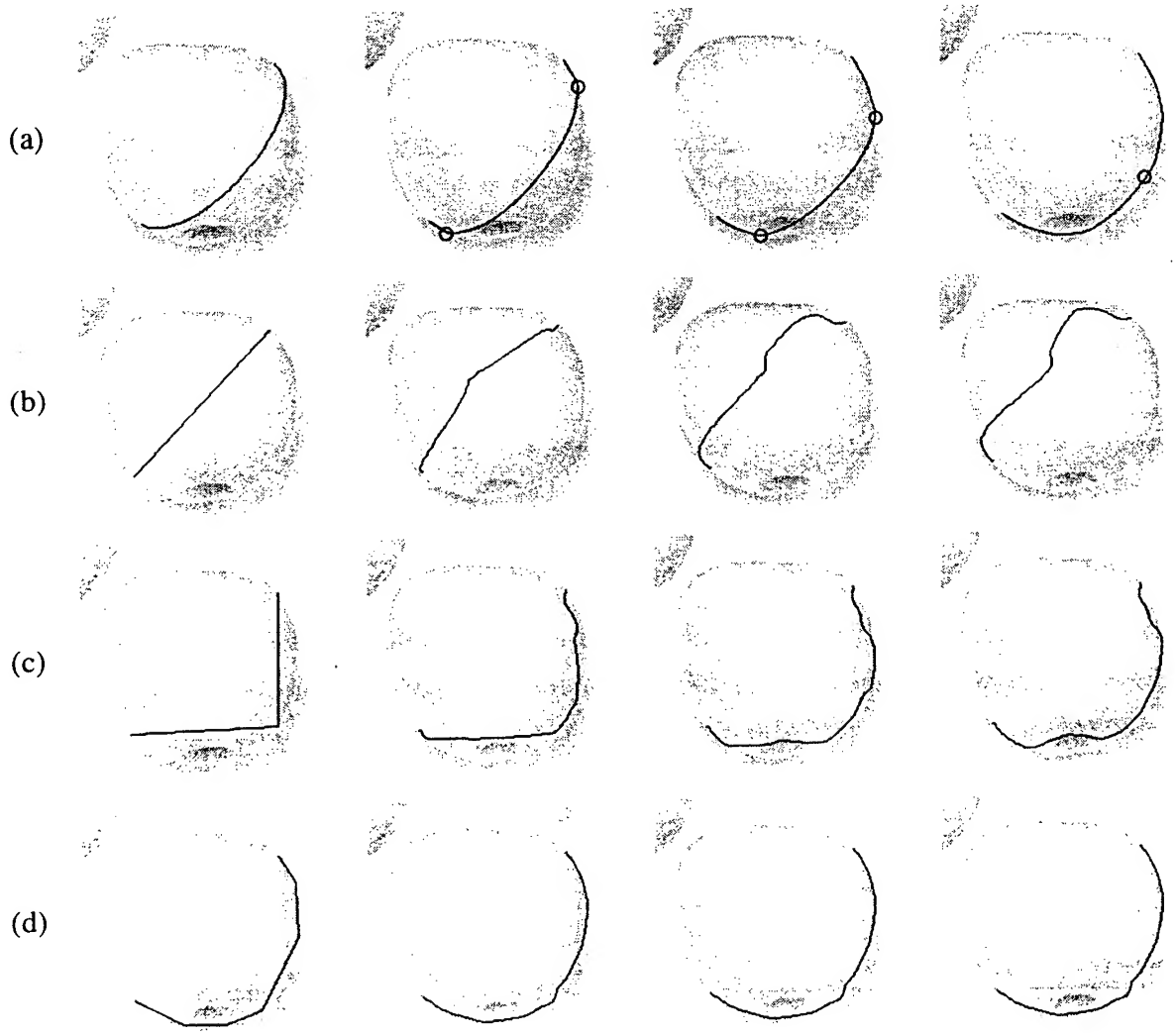


Figure 20: Compared behaviors of ziplock and traditional snakes on an image of an apple. The rows show the initialization and three stages of the optimization process for (a) a ziplock snake, (b) a traditional snake with the same initialization, (c) a traditional snake with three vertices, (d) a traditional snake with seven vertices.

4 DISCUSSION

We discuss and compare traditional and ziplock snakes with regard to initialization and contour following capabilities. Let us recall the basic assumption underlying the mathematical theory of deformable elastic models. The formalism is developed for small excursions from the stationary configuration for which internal and image forces are in equilibrium. The domain of validity of the mathematical model must be respected if convergence from an initial state to a representative solution is to be achieved. In traditional snakes, optimization is done for the whole contour simultaneously implying that the initialization has to be close to the equilibrium configuration along its whole length as well. By contrast, ziplock Snakes are initialized on single points lying on well defined edge segments and the optimization is initially confined to the close vicinity of these points. The small excursion assumption is thus satisfied at the beginning and remains fulfilled during the locking operation, at least as long as head and tail of the snake are progressively clamped onto well defined contour segments. Under these conditions, ziplock Snakes behave just like Growing Snakes. Once the contour becomes ill defined, however, Growing Snakes have to resort to stopping rules to decide whether to proceed or to stop since the contour is regarded too washed out to be traced further. Finding universally applicable rules has proved notoriously difficult[42, 43]. With ziplock Snakes, the decision to trace or interpolate faint parts of contour is delegated to the user and the astounding capabilities of the visual system. Picking two initial points means they are deemed connected, undisputedly an entirely pragmatic approach. We believe, however, it optimally combines the strengths of computer and human resulting in an easy-to-use interactive tool.

We turn to the contour-following capabilities which are different for traditional and ziplock snakes. The underlying reason is the different influence of the regularization term during optimization. In traditional snakes, regularization and image forces are always active from the beginning, and along the full length of the snake. As the snake evolves during optimization, the regularization term tends to force it to shrink and to prevent it from expanding to follow the contour as the image forces would require. This effect was recognized and addressed early in the history of snakes, e.g. by adding an expansion term to the mathematical model as in the balloon formalism[37]. In ziplock snakes the contour-following capabilities are governed by the interplay of passive and active snake segments and the movement of the force-boundaries. Recall that the image forces are turned-off between head and tail; the corresponding solution of the homogeneous Euler equations is known exactly and analytically while the active segments—for which the small excursion assumption is satisfied—are obtained via numerical optimization. After every displacement of the force boundaries the connecting force-free snake-segment is recomputed and grafted between the active segments. The length of this new passive part is essentially determined

by the updated (force-) boundary conditions; it can be shorter or longer than at the previous step. Premature shrinkage of snake-segments far away from equilibrium and thus, strictly speaking, still outside the domain of validity of the mathematical model will occur for poorly initialized traditional snakes while ziplock snakes will successively develop the proper length.

The examples below illustrate this behavior. We use synthetic and real images of tutorial character to compare the traditional snakes with our ziplock snakes and show that initialization of the former must typically be much closer to the desired answer to achieve comparable results. For a fair comparison, we use the same tension and rigidity parameters, α and β as defined in Equation (18), for both kinds of snakes.

Figure 18 shows three sequences of snakes evolving on a synthetic image. The first row (a) illustrates the behavior of our model. The ziplock snake can always be initialized with only two points. During the iteration, the force boundaries move along the contour until they meet. Sequence (b) illustrates the deficient behavior of the traditional snake initialized in exactly the same way. The third sequence (c) shows the traditional snake now initialized by a polygon with five vertices. These additional points are needed for traditional snakes to succeed. The ability of ziplock snakes to outline cavities and to distinguish between nearby objects is further illustrated by Figure 4.

Figure 20 compares the performance of ziplock and traditional snakes, using an image of an apple. Although the object looks quite simple, the segmentation problem is surprisingly treacherous. Shadows and illumination effects produce edges having nothing to do with the apple's outline. Sequence (a) shows the initialization and optimization of a ziplock snake. The two end points are selected on the apple's outline and the snake eventually deforms to the correct shape, even though, depending on the choice of α and β, the initial guess may be very far from the final answer. Sequence (b) illustrates the bad performance of the traditional snake model initialized using only the same two end points: the snake gets stuck on the bright specular reflection on the apple's skin. In sequence (c), the initialization is closer to the desired answer but the snake is still attracted by other salient features near the contour and fails to outline the apple. To achieve this result using a traditional snake, one must supply the almost "correct" polygonal approximation of Sequence (d). It is so close to the desired answer that only slight smoothing occurs during the evaluation, without any real position adjustment. In this case, the traditional snake acts only as a regularization tool smoothing out the first-order discontinuities of the initialization.

5 RESULTS

The contour-following capability of Ziplock Snakes and Growing Snakes [42, 43] makes them especially useful for segmenting scenes where the separation of nearby features is necessary. Figure 21 illustrates, using the same apple image as before, how easily the contour of the apple and its shadow can be separated. The first block of two images depicts the extraction of the apple's contour. To outline the shadow as shown in the second block, the initial points simply have to be moved into its vicinity.

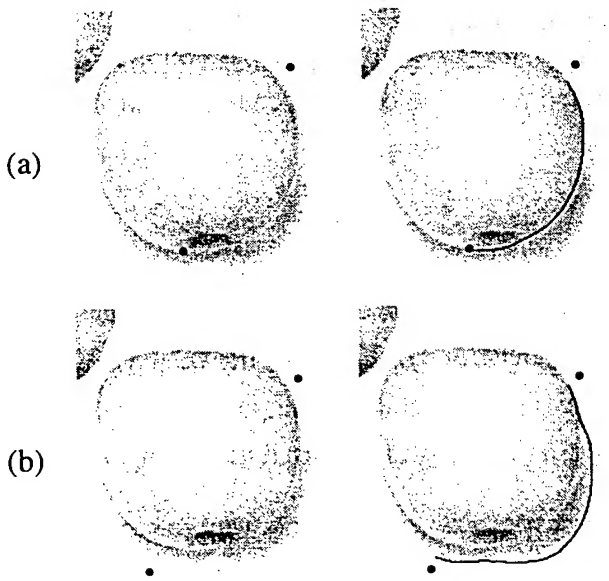


Figure 21: Detection of different image features by ziplock snakes. (a) Detection of the apple's outline. Some liberty can be used in placing the upper right initial point since there is no danger of ambiguity in the automatic adjustment to nearby image structures. More precision is required for the lower left initial point in order to avoid confusing apple and shadow boundaries. (b) Detection of the projected shadow; the lower left initial point is now closer to the shadow boundary.

Figure 5 shows that ziplock snakes can be used to delineate roads in an aerial image by using very distant end points. Note, however, that our snakes can still become confused in the presence of junctions. In the case of the curve drawn in the bottom left corner of Figure 5(b), there is a secondary road that leaves from the main one and traps the snakes in an undesirable local minimum. In practice, when such problems arise, our interface allows the user to add a new point in the middle of the curve, thereby splitting it into two snakes.

Another mechanism for the user to interact with snakes and fix problems such as the one described above is to allow him to use a mouse-controlled cursor to nudge the snake off the

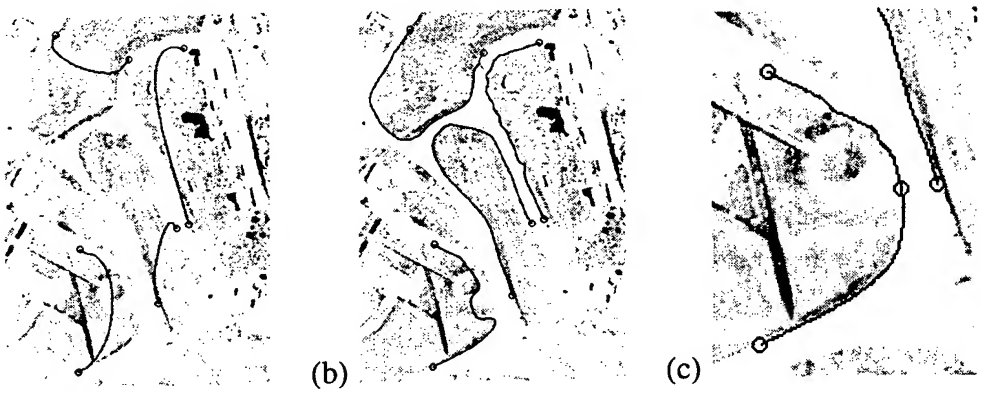


Figure 22: Outlining roads in an aerial image. (a) Ziplock snake initializations. (b) Final results. All the road edges are correctly outlined except the bottom left one. (c) The erroneous result is corrected by adding one new control point.

unintended feature. Alternatively, we have extended our formulation of the snake equations to include a term related to the length and direction of the vector from each snake vertex to the mouse-cursor. In this way, we have simulated a magnetic force between the cursor and the snake, which allows the user to push the curve in a desired direction.

To segment more complex shapes we need a simple way to sequentially define end points for adjoining open snake fragments. We have implemented an interactive initialization tool that starts the contour-finding process starts as soon as the user has clicked on two initial points. Sequentially adding salient points will then extend the initial segment under immediate visual control in a sort of contour-following rubber line process. This mechanism can also be utilized to detect closed contours since the boundary conditions can be shared between neighboring snakes to guarantee a smooth transition between them. Figure 23 illustrates this process on the segmentation of a corpus callosum in an MR image. Note, how the correct outline is found by the second segment in spite of its crossing over a wrong contour during the optimization phase.

Following [6], we have implemented a tool for interactive road delineation, called *ribbon snake*. The model vector $\vec{v}$ is augmented by a third component, the varying width $w(s,t)$ of the road. For the new model $\vec{v}(s,t) = (x(s,t), y(s,t), w(s,t))^T$ the expression for the deformation energy (18) still holds. The width is subject to similar "tension" and "rigidity" constraints as the two coordinate components. The ribbon forms the center of the road, while the assigned width defines two curves that are the actual deforming road boundaries. Note that the image information is taken into account along these two curves only. Figure 24 depicts the delineation of complete road boundaries using the ribbon snake model.

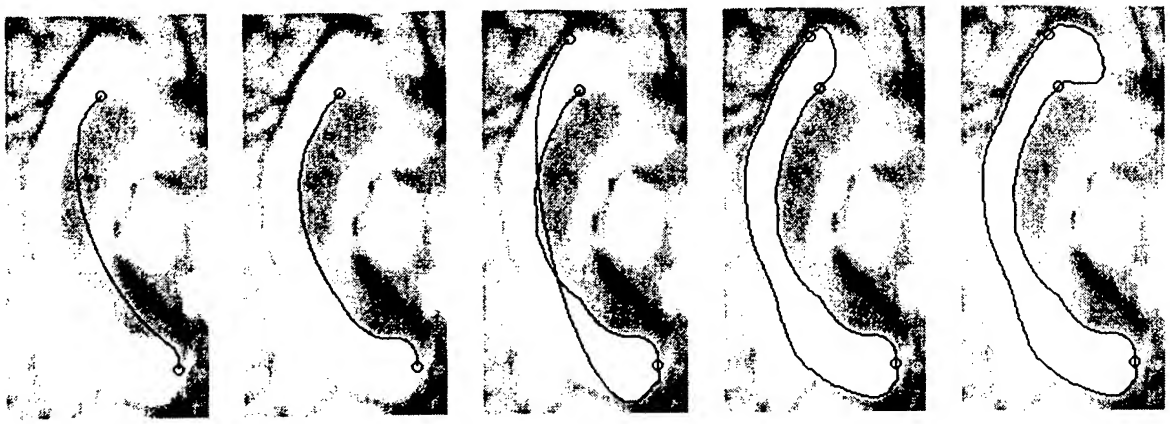


Figure 23: Segmentation process of a *corpus callosum* (MR image slice turned by 90°). The closed contour is built up by three open snakes linking three points supplied sequentially by the user and denoted by circles. In the middle picture, the contour initially overlaps the wrong edge; this has no adverse effect because, initially, its middle part is inactive.

6 CONCLUSION

We have proposed a snake-based approach to semiautomated delineation that allows a user to outline an open contour by specifying only very distant end points and allowing the computer to propagate edge information from the extremities toward the center. Our approach presents three main differences from more traditional snake implementations:

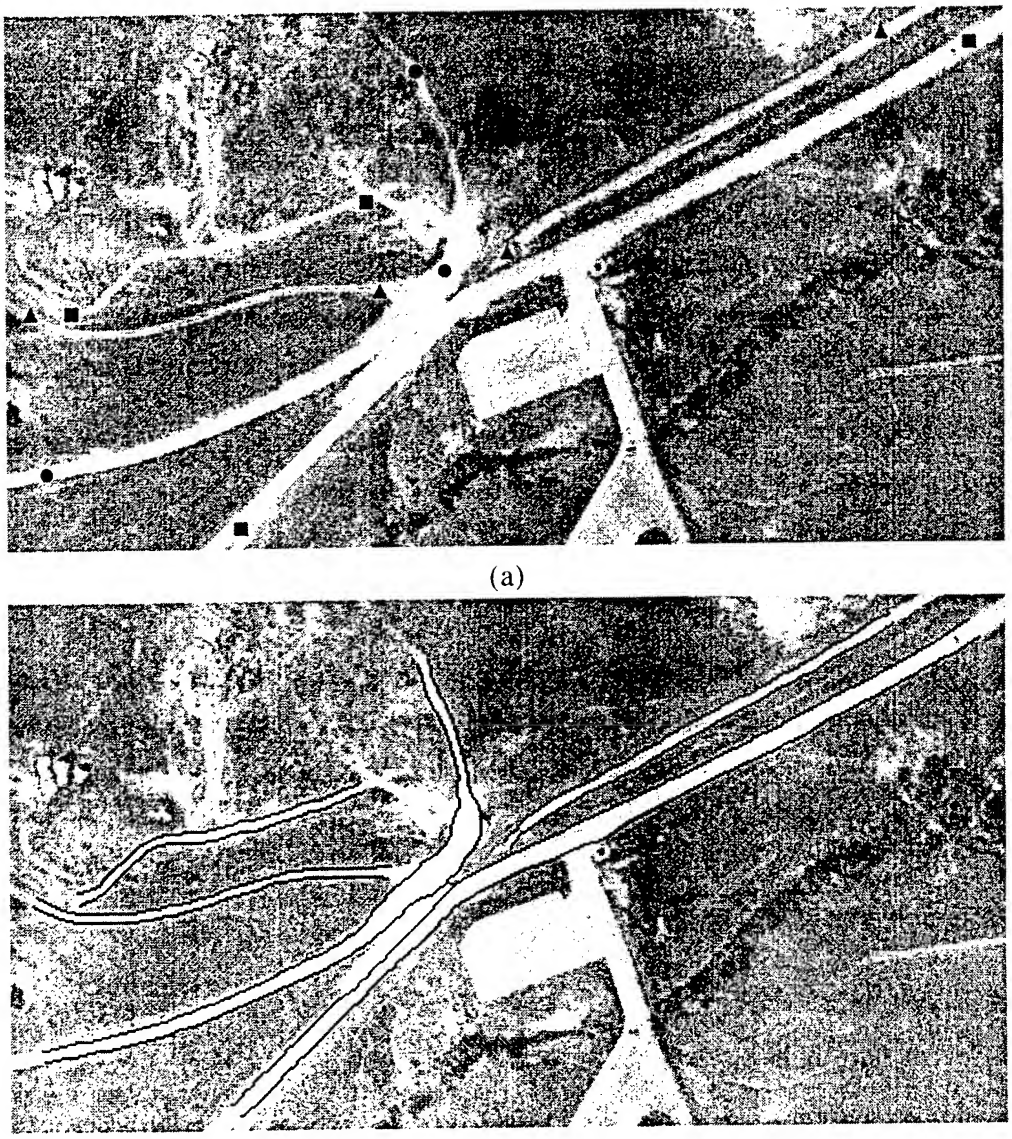
- It derives an appropriate set of boundary conditions from the image in order to constrain the optimization process.
- It uses a schedule that allows the snake to take image information into account first only near its extremities and then, progressively, toward its center.
- It allows the snake to grow or shrink by recomputing the force-free solution of the Euler Equation each time the force boundaries are moved.

Combining these ideas yields excellent convergence properties for the snakes and reduces substantially the probability of their getting trapped into an undesirable local minimum, even for initializations that are very far away from the desired answer.

Clearly, the motivation for deriving both approaches is identical. We both hope to significantly improve the predictability and convergence of the traditional snake method, and in both cases we have decided to adopt a local adaptation technique in order to deform the snake model to keep it

"on" the preselected contour. We see at least two clear advantages for the present approach. Viewing both systems as differential equation solvers, the Growing snakes are clearly formulated as an initial value problem, while the Ziplock snakes solve a boundary value problem. With essentially the same underlying differential equation, we expect a boundary value solution to be significantly more numerically stable. Secondly, the decision procedure adopted by the Growing snakes model to determine when it has extended beyond the "actual contour" is both complex (too complex to describe in this limited space) and arbitrarily tuned, with a number of adjustable thresholds. By contrast, there are only two decision procedures in the Ziplock snake process, the first to decide when to move the force boundary, and the second to decide when the final fully active snake has stabilized. Both of these procedures are simply, easily understood and framed entirely in terms of local behavior of the dynamical system and its relationship to the image resolution. It is thus easy to understand the relationship between the thresholds (1/4 pixel and 1/10 pixel) and the behavior of the snake.

Of course, the fact that the Ziplock Snake's evolution is more constrained by the boundary conditions may be seen as a drawback–it is unable to discover the full extent of a contour. However, the proposed framework is clearly meant to be used as an interactive tool. The only constraint on the user is the fact that the initial points must be close to the same well-defined contour. This is a constraint that can naturally be explained to a photoanalyst, even if he has no knowledge whatsoever of image processing. In other words, we have proposed a natural initialization procedure that is completely in line with both the practitioner's task and the mathematical problem so that the "expert user" [36] of the original snake papers need not be that much of an expert anymore.



(a)

(b)

Figure 24: Delineating roads on an aerial image using ribbon snakes. (a) Several pairs of endpoints. (b) Final detected road segments. Note that the actual ribbon, the centerline of the road, is not shown in the image on the bottom.

C THE SITE-MODEL CONSTRUCTION COMPONENT OF THE RADIUS TESTBED SYSTEM

Author: A. Heller, P. Fua, C. Connolly, and J. Sargent
Published in the proceedings of the IU Workshop, February 1996.

1 INTRODUCTION

We have chosen to measure the amount of effort expanded by the human analyst by the number of mouse-clicks and the amount of mouse-travel required to achieve a desired answer. We feel that this is a better measure than, for example, actual computation times because it truly reflects the amount of human interaction and does not depend on the speed of the computer being used.

We first briefly describe the code instrumentation that was required to perform the experiments and then report our results.

1.1 Instrumenting RCDE

We have developed and installed code that captures low-level information from the RCDE user interface about individual actions taken by the analyst. Every mouse-motion associated with making adjustments to object parameters, and every mouse click is captured into an event history. The following information is recorded:

- Object Adjustment Events:
  - object ID
  - event start time
  - adjustment type, for example vertex-xy, vertex-z, or vertex-width
  - 2–D world ID
  - zoom level
  - sequence of time deltas and mouse-position deltas of the form (delta-t dx dy)
- Mouse-Click Events:
  - object ID
  - event start time

- event ID (for example: zoom-in, zoom-out, recenter, drop-z)
- 2–D world ID
- zoom level
- mouse 2–D world position

This event history is then summarized by a small number of meaningful numbers. Among them are

- number of mouse-clicks
- number of mouse-moves
- total distance mouse moved during adjustments
- total time in adjustment events
- total time in fine adjustments

We have also implemented a metric to estimate the precision of an extracted road by comparing the centerline of the extracted road to the ground truth centerline. For each vertex of the extracted road, the distance to the nearest centerline point of the ground truth centerline is computed. The data is reduced to the following two numbers:

- Mean vertex to ground truth distance
- Maximum vertex to ground truth distance

These seven numbers appear in the figures Section 1.2.

1.2 Experimental Results

We used images such as the 7000×7000 one shown in Figure 25 to perform our experiments and chose a set of road segments to be modeled as accurately as possible. We compared four approaches to road delineation:

Hand Hand-tracing using the RCDE interface but neither snakes nor road tracker.

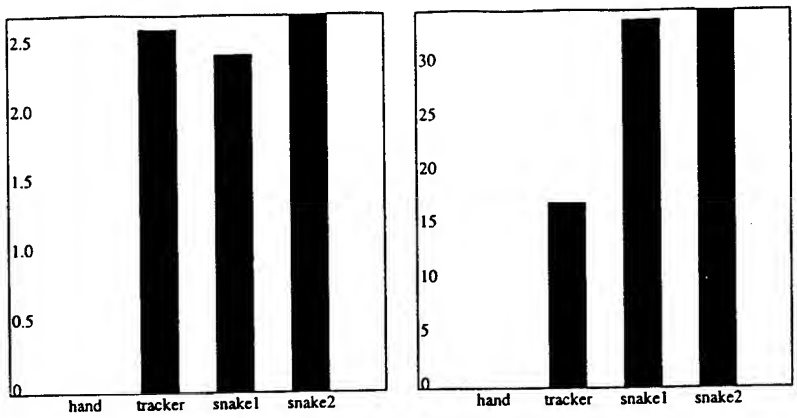
Figure 25: An image with two overlaid roads.

Tracker Using SRI's road tracker [32] to provide the initial sketch in the full-resolution image and then refining it using a ribbon snake [33].

Snake1 Sketching the road using the full resolution image and refining it using a ribbon snake.

Snake2 Sketching the road using a half-resolution version of the image, refining it using a ribbon snake first at half resolution and then at full resolution.

In all cases we used the system's default parameter settings and allowed the user to manually refine the automatically generated results to produce satisfactory delineations. The bar graphs that appear in Figures 26 and 27 are labeled "hand," "tracker," "snake1," "snake2."



(a) (b)

Figure 26: Distance to hand-entered roads. (a) Average distance difference (b) Maximum distance difference. Because the hand-entered results are taken to be the reference, the corresponding distances are zero.

We used the hand-traced versions of the roads as our references and the metric discussed above to evaluate the quality of the delineations produced by the three semiautomated approaches. As shown in Figure 26, the results are virtually indistinguishable in terms of average distance, whereas the "tracker" approach does better in terms of maximum distance.

Figure 27 depicts the amount of effort required by each approach, as measured by the number of mouse-clicks and the amount of mouse-travel. The tracker approach appears to be very effective and yields at least a sixfold improvement on all counts except the total number of mouse clicks. This occurs because starting and stopping each operation—automated sketching and snake refinement—require several clicks. This number could be drastically reduced by defining Common-Lisp methods that sequentially perform all these operations with a single mouse-click.

The snake2 approach is almost as effective but requires more effort to provide the initial sketch. This problem could alleviated by using Ziplock Snakes[34] instead of traditional snakes.

For all three semiautomed methods, however, a large portion of the human interaction goes into specifying the width of the road, as the current tools have no way of computing it. Therefore, methods to compute the width of a road given only its centerline would be extremely valuable and should be the object of future research. In short, by further improving the interface and developing a width-computing algorithms, we should be able to turn the current sixfold reduction of effort into a tenfold to hundredfold one.

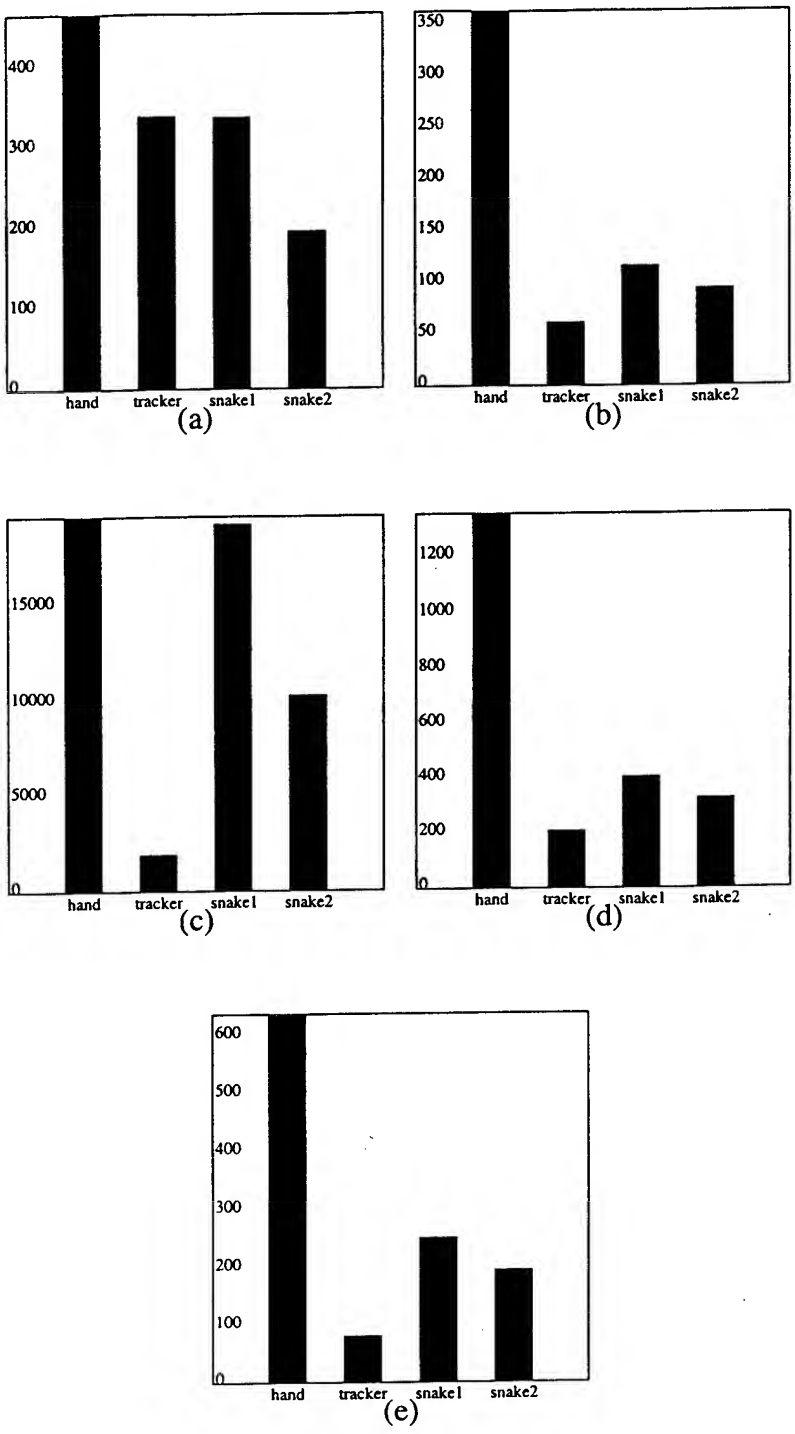


Figure 27: Amount of effort. (a) Number of object clicks. (b) Number of mouse moves. (c) Total mouse distance. (d) Total mouse move time. (e) Total mouse move time for fine adjustments.

D CONTEXT-BASED VISION

Authors: T. Strat, P. Fua and C. Connolly
To appear in *RADIUS: Image Understanding for Intelligence Imagery*, Oscar Firschein and Thomas M. Strat, editors, Morgan Kaufmann Publishers, 1997.

1 INTRODUCTION

Thirty years of research in image understanding (IU) have produced an enormous number of computer vision algorithms, many of which have demonstrated reliable performance at solving particular tasks in restricted domains. However, the development of computer vision systems that are reliable in more general circumstances has proved elusive [8]. It is in response to this situation that a framework is offered within which computer vision algorithms of specialized competence can be used and integrated with other such algorithms to produce a reliable vision system that operates more effectively in a broader context than any of its individual component algorithms can. Recently, other authors have also stressed the use of context to aid image interpretation [9, 10, 11].

The site model construction and editing system being developed within the RADIUS program comprises a large number of computer vision algorithms, each tailored to accomplish a particular task under a particular set of circumstances. The goals of these algorithms may overlap or even duplicate each other's goals, but the assumptions that they make and the data with which they operate will often differ. It is important that a user of the system not be required to understand the IU technology to take advantage of the IU algorithms. For this to happen, the system must be able to automatically select suitable algorithms and parameters to solve particular IU tasks. Within RADIUS this concern is paramount because of the need to put advanced IU technology in the hands of casual users, specifically the image analysts. Furthermore, the need to provide context information disciplines IU algorithm designers by forcing them to exercise their algorithm thoroughly to provide the required context information.

Our strategy for integrating such a collection of computer vision algorithms is to represent explicitly the assumptions made by each algorithm, and to use the context of the present task to select the most appropriate algorithms for solving that task. By doing so, we seek to avoid the source of many failures of computer vision techniques: the employment of an algorithm outside the bounds of its intended domain of competence.

This strategy is based on prior work in context-based vision at SRI [12, 3, 13]. This work resulted

in the CONDOR architecture that we describe first. We then describe the context-based architecture (CBA) that we have implemented within the Radius Common Development Environment (RCDE) and show that it constitutes an attractive framework for organizing a site model construction system using many algorithms that extract different features under different circumstances.

2 THE CONDOR ARCHITECTURE

In a context-based vision architecture, the development of general-purpose visual capabilities is attempted by assembling large numbers of special-purpose algorithms. Their invocation and the interpretation of results are mediated by the consideration of contextual information.

The image understanding system CONDOR was designed along these lines to serve as the perceptual architecture for a hypothetical outdoor robot [51]. Given an image and a possibly extensive spatial database, and a world model describing the robot's environment, the system is to analyze the image and to augment its world model. CONDOR's recognition vocabulary consists mainly of natural objects such as trees, bushes, trails, and rocks. Because of the difficulty of recognizing such objects individually, CONDOR accepts an interpretation only if it is consistent with its world model. CONDOR recognizes entire contexts, rather than individual objects [12, 52].

By making explicit the built-in assumptions inherent in computer vision algorithms, the CONDOR architecture allows context to influence the recognition process. Rather than employing a hard-wired control structure, the invocation of all algorithms is governed by context.

CONDOR associates a data structure called a *context set* with each IU algorithm. The context set identifies those conditions that must be true for that algorithm to be applicable. Efficient and effective visual recognition can be achieved only by invoking IU algorithms in those contexts in which they are likely to succeed.

Formally, a context set is a collection of context elements that are sufficient for inferring some relation or applying some algorithm. A *context element* is a predicate involving any number of terms that refer to the physical, photogrammetric, or computational context of image analysis.

Each algorithm has an associated context set, and is invoked only if its context set is satisfied. A context set is considered to be satisfied only if all its context elements are satisfied. As an example, consider a simple operator that extracts blue regions to find areas that could be labeled "sky." A context set for this operator might be

{ image-is-color, camera-is-horizontal, sky-is-clear, time-is-daytime }

The blue-sky algorithm would be unreliable if it were employed in anything but this context.

Other blue objects might also be detected in this context, but they are eliminated later by other contextual constraints, such as image position with respect to the horizon.

2.1 Context Sets

Context sets are used to specify the conditions that must be met for a given algorithm to be applicable. The context set can also specify the conditions that must be met for a given parameter setting to be useful. For example,

MBO(closed-curve, rectangular-corners, manual-entry, gradient-descent)

specifies the parameters for a model-based optimization (MBO) algorithm[1] that could be used to extract roof boundaries under some circumstances. The following context set encodes conditions that must be met for the successful employment of that algorithm to extract roofs.

{ **image-is-bw, image-resolution** $\leq$ 3.0, **interactivity-is-semiautomated** }

This context set gives the requirements that must exist for the MBO algorithm to be applicable and it specifies the suitable parameter values. In the previous example for detecting roofs, the parameters were specified as having a closed-curve topology, an objective function preferring rectangular corners, initial boundary provided by manual entry, and the use of a gradient-descent optimization procedure.

In practice, a large number of context sets governing the application of MBO algorithms as well as other algorithms could be constructed and used to implement a cartographic feature-extraction system suitable for site-model construction. It is clear that such a collection could be unwieldy and difficult to maintain. A more structured representation of the context-set concept is needed.

2.2 Context Tables

One alternative representation for context sets is the *context table* — a data structure that tabulates the context elements in a more structured fashion [53]. An IU algorithm is associated with each row in the table; each column represents one context element, as shown by Table 3.

The context table is equivalent to a collection of context sets. Conceptually, it provides a more coherent view of the contextual requirements of related algorithms. Applicable algorithms are selected by finding rows for which all conditions are met. One drawback to the table representation is its potentially large size. Each algorithm may require many rows to capture the contextual constraints of its various parameter combinations. Its chief value is its organization of

[1] MBO is discussed in Chapter 3 of this book.

Table 3: An Example of a Context Table

	feature	interactivity	resolution	geography	algorithm
1	roof	semiautomatic	≤ 3 m	—	MBO(topology=closed-curve, obj-fn=rectangular-corners, init=manual-entry, opt=gradient-descent)
2	roof	manual	≤ 10 m	—	CME(primitive=closed-curve)
3	road	semiautomatic	≤ 1 m	hilly	MBO(topology=ribbon-curve, obj-fn=(smoothness(0.5),continuous, parallel), init=manual-entry, opt=gradient-descent)
5	road	semiautomatic	≤ 1 m	flat	MBO(topology=ribbon-curve, obj-fn=(smoothness(0.8),continuous, parallel), init=manual-entry, opt=gradient-descent)
7	road	manual	≤ 10 m	—	CME(primitive=open-curve)
8	road	manual	≤ 1 m	—	CME(primitive=ribbon-curve)
9	road	semiautomatic	≤ 2 m	—	ROAD-TRACKER (control=bidirectional-search, init=manual-entry)

contextual information for knowledge-base construction.

2.3 Context Rules

A third alternative for representing context sets is to encode them as production rules whose antecedent is the context set, and whose consequent is the applicable algorithm. For example,

{ **image-is-bw, image-resolution** ≤ 3.0**, interactivity-is-semiautomated** } $\Longrightarrow$ **MBO(closed-curve, rectangular-corners, manual-entry, gradient-descent).**

One advantage of encoding the constraints as rules in a logic program is that using the logic program interpreter eliminates the need to devise special machinery to test satisfaction of context sets. The unification mechanism of the logic program provides a generic mechanism for matching the constraints embodied in a rule to the current context. Context rules can be more compact than the equivalent context table because additional predicates can be introduced to capture common context elements that appear in multiple rows of the table.

Whatever representation is chosen, it is clear that context sets can encode the information necessary to make intelligent choices of IU algorithms to solve particular tasks. Furthermore, they

can be employed in either direction. In the forward direction, the context sets are used to find applicable algorithms. In the opposite direction, the sets can be used for several purposes, including the selection of images on which to invoke a given algorithm.

While CONDOR has demonstrated a significant capability for recognizing natural objects in ground-level outdoor imagery, perhaps its more enduring contribution lies in its context-based architecture, which offers a design methodology with broad applicability. For example, CBA is an attractive framework for organizing a site model construction system using many algorithms that extract different features under different circumstances.

3 CONTEXT-BASED ARCHITECTURE: THE HUB

We now turn to the use of CBA in a 3–D site modeling system. We have implemented a system known as the HUB. The HUB provides a framework for semiautomatic selection of *appropriate* algorithms for certain task and site combinations. One of its primary goals is to prevent algorithms from failing under circumstances for which they were not designed. In addition to algorithm selection, the HUB can be used to select various other elements of context in an IU task. For example, a user can query the HUB for those images or feature types that are appropriate for a given algorithm.

A prototypical example used in this document is road detection: Lynn Quam's road tracker [54] works well on clearly defined roads at relatively high resolutions. For low-resolution images, where roads appear as thin curves, it might be more appropriate to rely on Ziplock Snakes [34]. Hence, the examples of HUB integration and usage given here employ these two algorithms, allowing us to explore some of the issues that arise.

The HUB has been implemented as a Prolog-based query system. The use of Prolog is motivated by its flexibility. Rules can be added and removed on-the-fly, and queries can be made on any number of variables. We use the version of Prolog defined in the book entitled *On Lisp* [55]. The Prolog engine has been modified to allow "hooks" to be executed at each inference. This allows the HUB to keep track of successes and failures, and is intended to provide the user with useful feedback on the inference process: For example, if no algorithm strictly satisfies the user's requirements, are there any which come close, say by one predicate? Based on the task to be performed, the HUB can query its database to determine which algorithm is suitable for applying on a given image, or for that matter which images or features are suitable for a particular algorithm. In addition, Prolog is used at the interface level to maintain the consistency of user interactions. Whenever the user selects options, the Prolog engine is invoked to ensure that the

```
(:algorithm ?name ?params)
     ?name:  The name of a lisp function
     ?params:  A list of the parameters to the function

(:task ?name ?feature ?operand)
     ?name:  A generic task name, such as :extract, :refine
     ?feature:  A feature type (eg, :building, :road, :functional-area)
     ?operand:  An object to be used, an image, a region, or other designation
           of a location where the task is to be accomplished
```

Figure 28: Documentation of compound terms.

```
(solves ?alg ?task)
     ?alg is an algorithm that solves the task ?task under some conditions

(invoke ?alg)
    ?alg is an algorithm that could be invoked to solve the given task
           in the current context
```

Figure 29: Documentation of some predicates.

```
(a)
; (invoke ?alg)
;     ?alg is an algorithm that could be invoked to solve the given task
;      in the current context.
(<- (invoke ?alg)
         (desired-task ?task)
         (solves ?alg ?task)
         (desired-interactivity ?int)
         (interactivity ?alg ?int)
         (desired-accuracy ?acc)
         (accuracy ?alg ?acc) )
(b)
; (solves ?alg ?task)
;      An algorithm that solves a task using an image,
;      also solves the task for the site of the image.
(<- (solves ?alg (:task ?name ?feature ?site) )
         (image-site ?image ?site)
         (solves ?alg (:task ?name ?feature ?image)) )
```

Figure 30: Documentation of some of the top-level rules.

user interface is in a state consistent with the HUB rule set.

3.1 Terms

Before describing the context-based architecture in detail, we introduce two concepts that are central to its operation: algorithms and tasks, as outlined in Figure 28.

Algorithms and tasks are represented in the HUB rule set as 3-tuples whose elements can be constants or variables. These tuples, in turn, are used in Prolog rules. As inferences are made, the variables within these tuples can be unified (that is, constants can be consistently substituted for these variables in an inference step) to produce *instantiations* of particular algorithms or tasks. Algorithms and tasks are normally coupled within the same rule so that successful unification produces a consistent algorithm/task pair.

An *algorithm descriptor*, represented by a 3-tuple, designates a piece of code that implements a computer vision technique. The first element of the tuple is the constant `:algorithm`. The second element is the LISP symbol that is the name of the algorithm. The third tuple element is a list of context-dependent parameters. The parameters will typically include an image, class of the LISP representation for the object, and type of feature to be extracted. For example, this tuple

```
(:algorithm road-tracker (fh-image-2 CME::3D-RIBBON-CURVE :road))
```

indicates that the road tracker algorithm is to be invoked on the image named `fh-image-2`, creating an object of class `3d-ribbon-curve`, and that it can be applied to find any road-like object.

A *task descriptor* is a tuple indicating an operation that can be performed. The first element of a task descriptor is the constant `:task`, followed by the name of the task, the type of feature that is desired, and an image. An example task is

```
(:task extract wide-road ?image).
```

Here, the task is to extract a wide road. The variable `?image` is left unbound, since this task description is used in Prolog as part of a larger tuple containing the algorithm descriptor. When successfully unified, the presence of an `?image` variable in the task and algorithm descriptors ensures a match between the algorithm and task descriptions, and results in an actual image to be used in the task.

3.2 Vocabulary

The first step in specifying HUB rules is to define a suitable vocabulary for rules. The vocabulary is used to describe images, sites, features, algorithms, and user preferences, to allow the HUB to select algorithms to apply in a particular task.

There are four main categories in the vocabulary:

- Algorithm
- Image
- Site
- Feature

Although the HUB itself uses Prolog, algorithm designers are not required to specify their rule packets in Prolog. For inclusion in the HUB, each algorithm should be accompanied by a set of basic declarations of the properties of the algorithm, followed by one or more lists of image and site conditions under which the algorithm will exhibit robust behavior.

Algorithm Vocabulary The algorithm vocabulary is used for basic declarations of algorithm properties. The algorithm designer should supply as many of these as is practical:

- `time` – estimate of running time (algorithm speed)
- `memory` – estimate of memory usage (kilobytes)
- `task-type` – e.g., extract, refine, or level of required initialization
- `interactivity` – e.g., manual, semiautomatic, automatic
- `accuracy` – e.g., coarse, fine

Image Vocabulary At present, the image vocabulary contains the following image properties:

- `dimensions` – image dimensions (numbers)
- `time-of-day` – time of day and date
- `imaging-geometry` – azimuth, elevation (degrees)[2]
- `registration-error` – (number)
- GSD – ground sampling distance – pixel width in meters at ground level (number)

[2]Intended for algorithms whose performance differs under nadir and off-nadir conditions.

- `footprint` – ground area covered by the image (number)
- `dynamic-range` – difference of maximum and mininimum pixel values (number)
- `cloud-cover` – percentage of cloud cover (number)
- `insolation` – INcoming SOLar radiATION W/m^2 (number)
- `spectrum` – portion of spectrum imaged (lo, hi, numbers)
- `snow-cover` – percentage of snow cover in the image (number)
- `albedo` – percentage of light reflected (number)

Site Vocabulary A number of site-specific properties form their own vocabulary:

- `facility` – type of facility
- `layout` – site layout (e.g., regular, random)
- `direction` – dominant direction (degrees from north)
- `vegetation` – percentage of site area
- `buildings` – percentage of site area
- `terrain` – flat, hilly, rugged, mountainous
- `terrain-quality`:
  - fake-horizontal
  - fake-planar
  - DTED-1
  - DTED-2
  - Hi-Res
- `georeferencing` – absolute, relative, none

Feature Vocabulary This vocabulary is used to describe the kinds of features that can be extracted or refined by the algorithm:

- `feature-type` – name of the feature type (e.g., road, building).
- `feature-vis` – percentqage of the feature that is visible.
- `feature-width` – width in pixels of the feature.
- `feature-length` – length in pixels of the feature.

In addition, an "interface vocabulary" expresses certain predicates, such as `desiredaccuracy`, `desired-feature-visibility`, and `desired-feature-2d-width`, in terms of the current state of the HUB user interface. Rules defining these predicates contain calls to LISP that access (and possibly change) the interface state. This interface rule set is the means by which the user's selections can be checked for consistency.

3.3 Predicates

The two most important predicates used in the context-based architecture are `solves` and `invoke` (Figure 29). The clause `(solves ?alg ?task)` is true if the indicated algorithm can be used to solve the indicated task. Here, `?alg` is an algorithm descriptor, and `?task` is a task descriptor, both as described above.

The clause `(invoke ?alg)` is true if the indicated algorithm is applicable to solving the current task. It is the truth of this predicate, and the associated binding of `?alg`, that the context-based architecture is expected to compute.

3.4 Rules

The primary rule governing the operation of the HUB is shown in Figure 30(a). It specifies the conditions that must be satisfied for an appropriate algorithm to be found to solve the currently desired task in the current context. In words, this rule states that an algorithm to be invoked must be capable of solving the desired task, and that its level of interactivity must be what is desired and its accuracy must meet the desired accuracy. Assertions of the clauses `(desired-task ?task)`, `(desired-interactivity ?int)`, and `(desired-accuracy ?acc)` are derived from the state of the user interface (i.e., are ultimately selected by the user).

```
(a)
; Chain rule:
; A task can be solved by splitting it into two parts.

   (<- (solves (:algorithm compose (?alg1 ?alg2) ) (:task :extract ?feature2
?image) )
          (solves ?alg1 (:task :extract ?feature1 ?feature2))
          (solves ?alg2 (:task :extract ?feature2 ?image)) )

(b)
; The interactivity of the composition of two algorithms with the
; same interactivity is the same.

   (<- (interactivity (:algorithm compose (?alg1 ?alg2) ) ?int)
          (interactivity ?alg1 ?int)
          (interactivity ?alg2 ?int) )

(c)
; If either algorithm is manual or semiautomatic, the composition
; must be semiautomatic.

   (<- (interactivity (:algorithm compose (?alg1 ?alg2) ) :semiautomatic)
          (interactivity ?alg1 ?int1)
          (interactivity ?alg2 ?int2)
          (lisp (not (equal ?int1 ?int2)) ))


(d)
; The accuracy of the composition of two algorithms is the accuracy
; of the last algorithm

   (<- (accuracy (:algorithm compose (?alg1 ?alg2)) ?acc)
          (accuracy ?alg1 ?acc) )
```

Figure 31: The Chain Rule.

Several auxiliary rules, such as the one shown in Figure 30(b), add versatility to the control structure. This rule states that an algorithm that solves a task using a particular image also solves that task for the site captured by the image. For example, the task

```
(:task :extract :building fort-hood)
```

can be reduced to

```
(:task :extract :building fh-image1)
```

if `fh-image1` actually portrays the Fort Hood site.

```
; (road-tracker ?image ?class)
; The top-level lisp function to invoke the SRI road tracker algorithm.
;    ?image:  The image to be used
;    ?class:  The RCDE class to be instantiated
(<- (solves (:algorithm 'road-tracker ?image ?class ?feature )
            (:task :extract ?feature ?image) )
        (image ?image)
        (subtype :road ?feature)
        (rcde-class-of-feature-type ?class ?feature)

        (feature-visibility ?v)
        (lisp (and (numberp ?v) (> ?v 90)))
        (hub-feature-width-2d ?feature ?w)
        (lisp (and (numberp ?w) (> ?w 6)))
        (desired-feature-width-property ?c)
        (is ?c :constant)
        (desired-feature-sides (?lx ?ly) (?rx ?ry)) )

(<- (interactivity (:algorithm 'road-tracker ?image ?class ?feat) :semi))

(<- (accuracy (:algorithm 'road-tracker ?image ?class ?feature) :coarse) )
```

Figure 32: The rule packet for the Road Tracker algorithm.

Posing the query `(invoke ?alg)` triggers the search of a binding for `?alg` that solves the desired task, `?task`. If successful, the binding of `?alg` that is produced (through unification) is typically a compound term specifying an algorithm, the imagery to be used, and any parameters it needs. If no applicable algorithm can be found, the search terminates and the failure is reported.

3.5 Chain Rule

The use of the Chain Rule given in Figure 31(a) allows the interpreter to search for a sequence of algorithms that collectively solve the desired task. The rules in Figure 31(b), (c), and (d), describe how the interactivity and accuracy of a composition of algorithms relate to the properties of each individual algorithm.

3.6 Rule Packets

Introduction of a new algorithm into the system requires specification of a packet of rules that delimit its scope of applicability, its properties, and its assumptions. The primary means for this is the `solves` predicate.

```
(<- (solves (:algorithm 'ziplock-snake ?image ?class ?feature)
            (:task :extract ?feature ?image))
        (image ?image)
        (subtype :road ?feature)
        (rcde-class-of-feature-type ?class ?feature)
        (desired-feature-ends (?sx ?sy) (?ex ?ey))
        (lisp (and (numberp ?sx) (numberp ?sy) (numberp ?ex) (numberp ?ey)))
        (gsd ?image ?x)
        (lisp (and (numberp ?x) (>= ?x 3.0)) ))
```

Figure 33: The rule packet for the Ziplock Snakes algorithm.

For example, Lynn Quam's Road Tracker algorithm [32] seeks to follow the extent of a road by correlating successive cross sections starting from an initial seed point. It requires that the width of the road occupy at least 6 pixels. The rule packet governing the application of this algorithm is given in Figure 32. The first rule states that the algorithm:

1. can be used to extract any feature that is a subclass of type road
2. needs the feature to be more than 90% visible
3. requires more than 6 pixels for the feature width in the image
4. requires that the feature width be constant
5. needs the user to pick two sides for the road
6. will use an instance of an RCDE class suitable for modeling the desired feature type.

The remaining two rules specify that the road tracker algorithm is inherently semiautomatic (because it requires the user to provide initial seed points), and that the delineation accuracy that can be expected is qualitatively designated as "coarse."

The rule for invoking the Ziplock Snakes [34] is depicted by Figure 33. It is similar to the rule of Figure 32 except for the fact that the ground-sample distance must be larger than 3.0 meters, and that the user must supply endpoints. In practice, the HUB will therefore tend to use the road tracker in high-resolution imagery and the Ziplock Snakes in low-resolution imagery. In addition to the road-tracker and Ziplock Snakes, the HUB currently also has access to a ribbon-snake algorithm [6] and to the F* dynamic-programming algorithm [56]. Ribbon snakes are invoked on high-resolution, variable width roads, while F* is typically invoked on very low-resolution images.

3.7 User Interface

One of the important features of the HUB is its ability to maintain a consistent internal state at any point in the model construction process. Most user interactions with the HUB interface will trigger the Prolog system, which allows the HUB to fill in missing information if necessary, or to inform the user that certain choices are inconsistent. For example, if the user specifies that a particular road has a certain width, the HUB will determine the road subtype (e.g., highway, two-lane road) that is consistent with that feature width. In addition, every user interface item has a "guess" option, which invokes Prolog to find a suitable value for that interface item given the states of the other items. For example, one can ask the HUB for feature types that can be extracted by a given algorithm by first selecting an algorithm, and then selecting the "Guess" option in the feature type menu. Prolog is then invoked to query on the acceptable feature types for the specified algorithm.

The use of Prolog also allows the HUB to make queries on more than one variable. Using this feature, the HUB initializes itself by performing a general query of the rule set on *all* items in the user interface. The currently installed rule packets dictate suitable values for each interface item. When the HUB interface is presented to the user, any interface items that can be defaulted in this way will have values, the exceptions being items which require user input (e.g., sides of roads, selection of functional regions).

When enough information is present, the user need only hit the "Run" button, which will select the appropriate algorithm and start the requested task. In the case of road delineation, the user typically need only select endpoints, and possibly road sides, before selecting "Run", thereby invoking one of the four road delineation algorithms.

Once the task has been completed, the user is prompted for any editing (if necessary), and an accept/reject decision. If accepted, the object (with possible edits) is saved in the HUB feature set. Each object so created is also tagged with the state of the HUB at object creation time. This allows "construction by example", wherein the user can request that the HUB create a new object with the same parameters used to create a previous object. In this case, the HUB simply copies the relevant parameters from the older object into the interface.

Figure 34 illustrates the interaction of the user with the system. The first example, shown in Figure 34(a,b,c), deals with a road in a high-resolution image. Given the task description, HUB selects the Road Tracker algorithm. The system asks the user to supply start and end points and to indicate the road's width, and then finds it. The second example, depicted by Figure 34(d,e,f), demonstrates low-resolution road delineation: This time HUB selects the Ziplock Snakes

algorithm to trace the road. Note that, in both cases, the interface provides the user with an indication of the context and the parameters used.

4 CONCLUSION

The context-based architecture described was designed to enable the construction of large, reliable image understanding systems by integrating a collection of reliable, but specialized algorithms. Initial experiments indicate that progress has been made in this direction, but formal testing of even larger assemblages of algorithms is necessary to validate this belief.

Some of the known limitations of our current implementation are that

- There is no notion of likelihood of success. All algorithms are characterized as being either applicable or not. Incorporating machinery to manipulate probabilities or other uncertainty measures is possible, but ascertaining the likelihood of success of IU algorithms under many and varied circumstances could be difficult.

- Incorporating additional algorithms requires hand crafting the rule packets that govern their employment. Automated knowledge acquisition to automate this process is needed to alleviate the burden.

- The CBA already has the ability to find sequences of algorithms to solve a given task. However, it has no procedure to check the validity of intermediate results at runtime.

It is clear that the performance of an IU system employing the context-based architecture could also be attained by integrating the same computer vision algorithms via more traditional methods. However, the explicit representation of contextual constraints affords a number of additional benefits that would be lost to a purely functional integration. These benefits include

Task specification: The context-based architecture allows the user to specify the task to be accomplished, leaving the selection of specific algorithms to be decided by the system. For example, the user can state that he would like the system to construct a 3-D model of a building, and the system would decide which of several building-extraction algorithms would be most appropriate given the currently available imagery and auxiliary data. The user can make effective use of the IU system while possessing little knowledge of the capabilities and limitations of the individual computer vision algorithms.

Choosing parameters: The context rules are used to establish algorithm parameter settings, in the same way that they delimit the range of applicability. While computer vision algorithms can often compute their parameter settings from data at runtime, the context rules provide a uniform means for all algorithms to specify how their parameters are to be determined.

Integration: When building large systems in an evolutionary fashion, it can be difficult to add new capabilities without jeopardizing the integrity of existing capabilities. The modular decomposition of the context rules allows the developer to integrate a new algorithm by adding a packet of rules governing its use, without modifying existing code.

Choosing imagery: It is sometimes important to identify the imagery that is most likely to allow an algorithm to yield a desired result, rather than to choose an algorithm to run on a preselected image. The context rules already encode the information necessary to make this determination — they can be used to answer this question by fixing the algorithm and allowing the image to be a variable in the query. In fact, the context-rule base can be used to answer both questions simultaneously, finding the best combination of algorithms and images to satisfy a given task.

In summary, the HUB provides a "firewall" that relieves the user from any need to know the details of IU algorithm implementation and testing. The HUB rule packets define the proper contexts for applying different algorithms. This greatly simplifies the effort required by a casual user for site model construction. The likelihood of algorithm failure in performing a given task is reduced (and ideally eliminated). The user need not waste time experimenting with algorithms to determine their failure points. In addition, the HUB can serve a useful pedagogical role: Interaction with the HUB will give a user greater familiarity with the contexts under which different algorithms can be successfully applied.

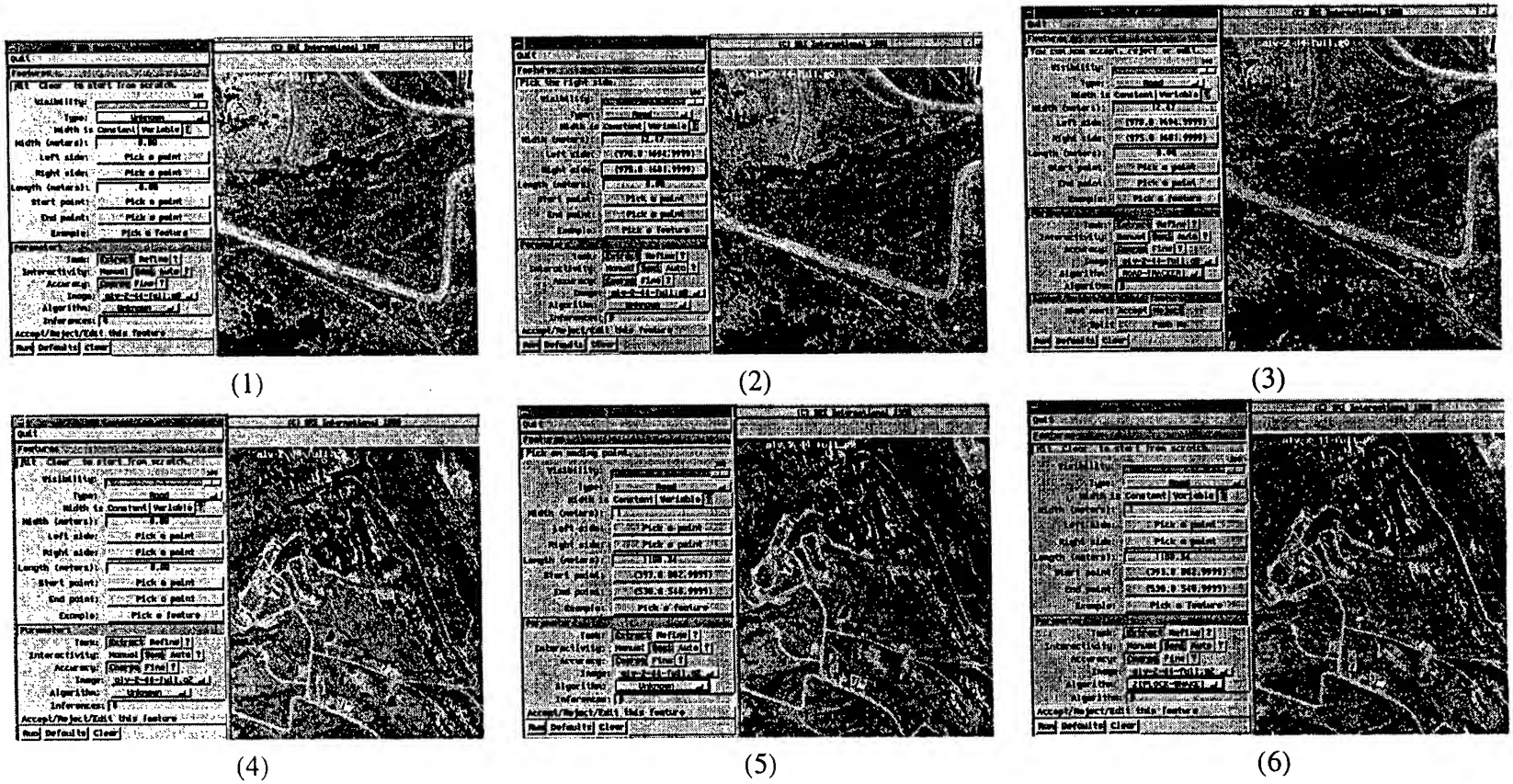


Figure 34: Interacting with the HUB: (a) The user is presented with the HUB interface containing defaulted values obtained from the rule set in a high-resolution context. (b) The user has selected the feature type (road) and the sides of the feature. (c) The HUB invokes the Road Tracker to place a road. (d) Another defaulted HUB menu in a low-resolution context. (e) The user has selected endpoints for the road. (f) The HUB has used the Ziplock Snake algorithm to place a new road.

E LEARNING CONTROL PARAMETERS OF A VISION PROCESS USING CONTEXTUAL INFORMATION

Authors: S. Houzelle, T.M. Strat, P. Fua and M.A. Fischler
Published in the proceedings of the ICPR Conference, October 1994.

1 INTRODUCTION

The research effort described here is an attempt to make computer vision systems more effective by endowing them with a capability to learn. Our philosophy has been shaped by the following principles, each of which has motivated some aspect of our approach:

Learning is essential — image understanding system designers cannot anticipate all possible situations that may arise in advance.

No matter how much effort is devoted to building and refining a knowledge base, there will always be limits to the breadth of competence provided. Even if it were possible to construct a knowledge base sufficient for supporting the entire range of anticipated tasks, the required effort and expense make it infeasible to do so in all but the most limited applications. Effective mechanisms for enabling a system to acquire its expertise over time can have a significant impact on our ability to construct systems that become more (rather than less) competent as they age.

A vision system should improve its performance through experience.

Rather than analyzing images in isolation and throwing away the results, a vision system should interpret an image in the context of what it already knows about the scene. In addition, the results of its interpretation should augment its knowledge of the scene and the extraction task, and the system should be able to use that information to analyze similar situations more effectively in the future.

An intelligent system should never be idle.

If an intelligent system has the ability to learn through experience, it might as well devise its own training examples to more fully exploit that ability. Rather than maintain a static knowledge base when the system is not otherwise engaged, it should concoct new situations or revisit previous

ones, invoke its repertoire of reasoning or visual capabilities, and update and reorganize its knowledge base according to the results.

While the accomplishment of any of these objectives is profoundly difficult, we nevertheless have endeavored to construct a framework that allows the exploration of a particular approach to learning that offers the promise of at least partial solutions. More specifically, our overall goal is to devise a practical computer vision system that can

- Recognize a class of objects in a set of related images
- Build an enhanced description of the environment from the sequence of images it processes
- Use its enhanced description of the environment to improve its recognition capabilities

1.1 A Cartographic Application

Imagine a cartographic application in which the photointerpreter's task is to model all roadways within a given geographic area by extracting such features from a collection of overlapping overhead images. The photointerpreter is to be assisted by a partially automated system designed to support 3–D map construction.

The traditional interactive approach embodied in today's operational systems [3] can be described as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of specified objects. The user chooses the algorithm to be used to extract a particular object. A menu is provided containing the default values of parameters, which the user can override if he chooses. The algorithm with the designated parameters is then applied to the image(s) producing a resultant model instance. The user has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, or to choose a different algorithm.

Two of the problems that the user of such a system must face are the choice of algorithm and the setting of its associated parameters. These requirements mean that the user must have a fairly high degree of expertise with the algorithms to accomplish the extraction task effectively.

[3] e.g., RCDE, GLMX, DSPW, KBVision, Geoset

If, on the other hand, the system is itself able to learn how to select among its algorithms and to set their parameters through its experience with similar extraction tasks, it should be possible to reduce the need for operator expertise while improving efficiency at the same time.

The approach we propose and describe in this paper addresses the feature extraction task as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of specified objects. The user chooses the class of objects to be extracted and provides information about his task and the scene being analyzed. The system compares the extraction context to its prior experience with similar extraction tasks, to choose an appropriate algorithm and parameter settings for the current task. The algorithm with the designated parameters is then applied to the image(s), producing a resultant model instance. The user then has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, to choose a different algorithm, or to ask the system to provide an alternative selection of algorithm and parameters. The system updates its database of experience with the outcome for use in subsequent extraction tasks.

The focus of this paper is on establishing a foundation for machine learning in interactive image understanding systems — how can a computer vision system use its experience to improve its competence? The design and implementation of a widely used interactive computer vision system, the RADIUS Common Development Environment (RCDE), has been discussed elsewhere [57, 58]; this effort is intended to enhance the performance of RCDE through the addition of a learning component. We build upon the notion of a context-based vision system that has also been previously developed [59, 52].

1.2 An Approach to Learning Visual Parameters

The user in our interactive system is a critical component in the automated learning process, even though he is not necessarily aware of this role. The user provides the performance evaluation and correction feedback that is necessary for directed learning and avoids the need for a computationally infeasible trial-and-error approach. Our system takes advantage of the human review of its results to determine how successful it has been, to reinforce its successes, and to take corrective action for its failures.

Two different problems must be addressed. The first one is to find the best algorithm to run, and

the second one is to determine the best parameter setting for the selected algorithm in the given context. Only a few authors have addressed these problems, and all are concerned with finding the best strategy to accomplish a complex task necessitating several steps or subtasks. At each step and depending on the data, they pick the best procedure among all possible to perform the subtask. They focus on the choice of an algorithm, and typically are unclear on how they set the control parameters. In the domain of pattern recognition, we can cite the work of Draper et al [60, 61], which involves learning the best strategy to identify a certain type of object. A very interesting point about this work is that they are able to give an upper bound of the cost of each strategy [61]. We can also cite the OKAPI system [10], which more generally, can select the best image processing chain to perform a given task. OKAPI has been applied successfully in the domain of galaxy recognition.

Our approach has been guided by two points. First, our concern is quite different from the ones above, because we are considering only one-step tasks[4], and we assume we can have, in certain cases, only one algorithm to achieve the task. Thus, we are primarily interested by the choice of good parameter settings for a given algorithm.

Second, we believe that it is useless to choose a so-called best algorithm for a given task if we are unable to set the control parameters correctly for any set of data.

Thus, the scheme we adopt is first to find the "best" parameter setting for every algorithm available to achieve a specified task. Then we give a score to each parameterized algorithm, showing the chances of the algorithm to succeed with the particular parameter setting. Finally, we run the top-ranked algorithm.

Our approach is based on the use of contextual information. Many authors have used contextual information in image understanding systems [62, 63, 64, 53], but few have made the use of context a way to improve the performance of systems through experience and learning. We define *context* as any information that may characterize the task or input data given to a vision process. Thus, image resolution is part of the contextual information, as is the camera geometry, a priori scene knowledge, and the purpose of image analysis. The approach relies on the assumption that each "running context" (or simply context), consisting in a set of contextual values, can be related to an acceptable parameter setting with a minimum of ambiguity. This requirement implies a kind of continuity in both context and parameter spaces. This continuity can be expressed as follows:

Different contexts require, in general, different parameter settings while identical or close contexts would require a single or similar parameter setting(s).

[4]Note that this is not restrictive because if a task is more complex and requires several steps we can apply the procedure we have designed on each step of the task.

Using this hypothesis, our learning problem can be defined as a problem of generalization where the goal is to find how the use of one parameter setting for one context can be generalized to similar contexts, or more generally, nearest contexts.

In Section 2, the general scheme of the system is presented. We see that the various possibilities offered to the user make this scheme very flexible. This flexibility and the need for using various algorithms has implied certain constraints in the design of the system. We present them in Section 3. The principal problem that arises is the presence of both numerical and categorical context elements.

In Section 4 we focus on the retrieval problem which, given a new context, consists in finding the nearest contexts that are present in a data base. To find the nearest contexts, we propose a measure based on similarity between context element values. This measure deals with both numerical and categorical context elements.

This kind of approach has some computational limitations and limited learning capabilities. Thus, we have investigated another method based on incremental categorization, that places a new context into a category where nearest already encountered contexts can be found. This kind of generalization is very important to reducing the search time for similar situations. *Learning by observation*, and more particularly *conceptual clustering*, which is a type of learning by observation aimed at producing a classification for the observations, are well adapted to this categorization problem.

In Section 5, we present different aspects of learning by observation and review some existing systems using this technique. We see the extent to which the constraints of our problem are satisfied by existing systems.

In Section 6, we focus on one particular method of conceptual clustering, and we enhance this method to deal with heterogeneous types of variables.

In Section 7, we present the learning update, consisting in updating with new examples the data bases where past experiences are stored, and eventually automatically learning a correct parameter setting in cases where the system was unable to provide one. This phase is essential to improving learning capabilities of the system.

In Section 8 and the Appendix, we describe the snake algorithm that we use to demonstrate our system's effectiveness. Snakes [5, 4, 65] are a very powerful technique for edge detection that integrate information from both photometric and geometric models in an optimization framework. More specifically, we show how our implementation allows the various parameters to be

context-specific as opposed to image-specific.

Finally, in Section 9, we present some experimental results; we show how our system makes the use of vision algorithms easier by reducing the required user expertise while improving his efficiency. We have implemented and tested an initial design and demonstrated successful performance. Our experiments involved the extraction of 68 features in four images of two different sites. These results are presented in graphical form where the effects of successful learning are clearly apparent.

2 GENERAL SCHEME

The system we have designed is intended to use computer vision algorithms to extract cartographic features from a set of imagery under human guidance. The system makes use of a collection of algorithms, and must select appropriate parameter values prior to each invocation of an algorithm. Choice of algorithm and parameter settings is to be made on the basis of contextual information.

In the following a *context element* defines a contextual variable. A *context* will be a generic term to define a contextual environment. Each context is characterized by a context vector regrouping context element values. A *parameter setting* is a set of parameter values needed to run an algorithm. A parameter setting is more formally represented by a *parameter vector.*

Let $\mathcal{A}(\mathbf{D},\mathbf{P})$ be a process that takes two kinds of information as input, data represented by a vector $\mathbf{D}$ (which includes specification of the task), and parameters represented by a vector $\mathbf{P}$. $\mathcal{A}$ gives a solution S as output. Suppose we have calculated some context information about the task and data represented by a context vector $\mathbf{C}$.

The primary performance criterion for a practical system for interactive feature extraction is its ability to generate a good result in a previously encountered situation while avoiding the repetition of past errors. Thus, it is necessary for the system to keep a record of its successes and failures. Operator review of the automatically generated results provides the necessary information in a natural way.

From these considerations, we derive the general scheme for our system, depicted in Figure 35. One data base (DB) is associated with each algorithm available to perform a given task. These data bases contain past experiences expressed by pairs ($\mathbf{C}$, $\mathbf{P}$) given the correct parameter setting $\mathbf{P}$ in the contextual situation $\mathbf{C}$. Given a new context vector $\mathbf{C}$, these data bases are used to retrieve the "best" algorithm $\mathcal{A}$ and a parameter vector $\mathbf{P}$. The process $\mathcal{A}(\mathbf{D},\mathbf{P})$ is applied and the

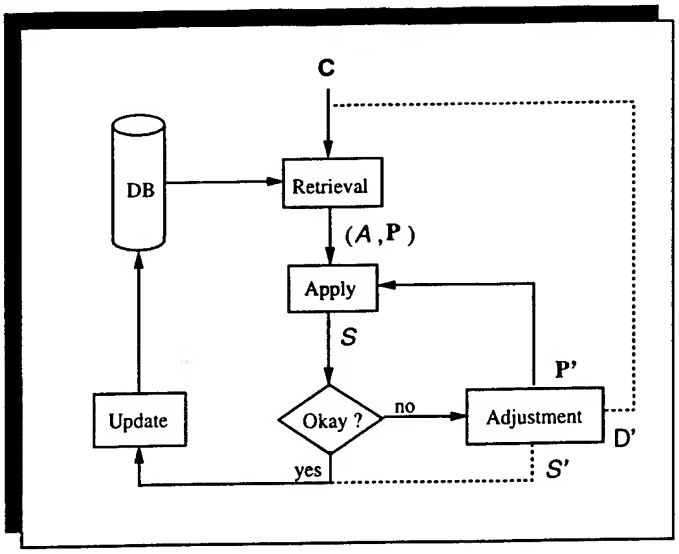


Figure 35: General scheme

user checks the validity of the result S. If S is acceptable, **P** is stored in order to update the parameter value probabilities in the data base. If $\mathcal{A}$ failed, the user has three adjustment options. The first one is to manually modify the parameters and to apply $\mathcal{A}$ again until it succeeds. The second option is to manually modify the result S so it becomes reasonably good. In this case, updating the data base consists in learning a correct parameter setting for the current situation. Because the solution S is available, the learning can be done automatically. This phase is usually performed after the session. Finally, the third option for the user is to modify some aspects of the input data **D**. In such a case, a new context vector is calculated and the procedure starts all over again. In this way, the system is able to improve its performance over time, as the likelihood that the system encounters a context similar to one in which it has successfully accomplished its task increases monotonically.

This architecture can serve as the foundation of a practical system for cartographic feature extraction, while affording a path to the creation of a vision system with very powerful constructs for learning.

3 APPLICATION DOMAIN CONSTRAINTS

The interactive nature of our system and the wide variety of data types it must deal with pose additional challenges to the design.

The most important constraint (or capability) we have is to be able to deal with different types of data. Both context elements and parameters can be either numerical or categorical. Table 4 shows

examples of some context elements that can be used. The *resolution* is a numerical context element with continuous values. On the contrary, the *task* has categorical (nominal) values. Finally, *desired accuracy* is a context with ordinal values – that is, symbolic values that can be related to a numerical discrete function (like *low* = 1, *middle* = 2, *high* = 3).

Table 4: Different types of variables handled by the system.

context element	type	possible values
resolution	numerical, continuous	0.1, 10, 100
task	categorical	road, building delineation
desired accuracy	ordinal	low, middle, high

Most of the existing systems found in the literature consider only one type of data. Few systems use both numerical and categorical variables, but consider numerical values as categorical. This is often acceptable with ordinal variables, but not with continuous numerical ones.

Employing numerical values is a major problem for generalization, because close values cannot be considered the same as completely different values. On the other hand, dealing with categorical variables can also be a real problem when similarities between values are required. We will see in Sections 4 and 6 two solutions to deal with all types of variables presented in Table 4.

A second constraint we have about variables is due to the fact that our system is interactive. One role of the user is to provide values for context elements that cannot be computed automatically. The interactivity is an opportunity to increase system performance. However, the user must not be annoyed with too many constraints. Thus, we have to take into account that the user can decide not to provide every context element values, and so that some context elements may have the value *"unknown"*.

The number of context elements we use as well as the number of parameters an algorithm has can be high. For instance, results shown later involve sixteen context elements and an algorithm with eight parameters. Spaces of context elements and parameters have a high dimension, and the expertise of the system cannot cover these spaces entirely. There are two consequences. First, the system should be able to learn in an incremental way. Second, because of high dimensions, initial learning may be poor and the learning method has to be able to deal with radical changes.

4 RETRIEVAL BASED ON SIMILARITY

Here, we focus on the retrieval module of our design as depicted in Figure 35. Given the context of a feature extraction task, we wish to identify the previously encountered contexts that are

"nearest" to it. The parameter settings associated with those nearest contexts will be used to choose the parameters for the current extraction task.

Using our hypothesis of continuity in context and parameter spaces, and given a new context vector $\mathbf{C}$, the retrieval problem can be more precisely defined as finding the nearest context vectors $\mathbf{C}_{j1},..,\mathbf{C}_{jn}$ present in the data base of algorithm $\mathcal{A}_j$, and providing parameter vectors $\mathbf{P}_{j1},..,\mathbf{P}_{jn}$ associated with these nearest context vectors. A score has to be associated with each $\mathbf{P}_{ji}$ to rank the parameter setting according to its ability to produce a good result. The top-ranked pair $(\mathcal{A}_j, \mathbf{P}_{ji})$ is finally provided to the user as having the maximum chance to succeed as it corresponds to the nearest situation already encountered[5].

We investigate a measure based on similarity to find the nearest context vectors $\mathbf{C}_{j1},..,\mathbf{C}_{jn}$ present in a data base. Information retrieval is a domain that has produced a large number of similarity measures to comparing two vectors (see [66] for an extended list). Let $\mathbf{u}$ and $\mathbf{v}$ be two vectors in a n dimensional space. An important family of pseudo linear similarity measures between $\mathbf{u}$ and $\mathbf{v}$ is defined by:

$$m(\mathbf{u},\mathbf{v}) = \frac{\mathbf{u}.\mathbf{v}}{N_1(\mathbf{u})N_2(\mathbf{v})} \tag{28}$$

where $\mathbf{u}.\mathbf{v} = \sum_{k=1}^{n} u_k v_k$ denotes the scalar product between $\mathbf{u}$ and $\mathbf{v}$, and N_1 and N_2 are two normalizing functions (generally $N_1 = N_2$). For example, the very popular *cosine measure* is defined with $N_1(\mathbf{u}) = \|\mathbf{u}\|$, and $N_2(\mathbf{v}) = \|\mathbf{v}\|$, where $\|.\|$ denote the l_2 or Euclidean norm.

This kind of ranking guarantees *under certain conditions*, that the less preferred objects are not ranked ahead of the preferred [66]. However, because of the possible categorical context elements in our problem formulation, this kind of interesting ranking function is not directly usable.

A heuristic solution to this problem is to use one of these ranking functions in a real valued space, transformed from the context space. The transformation we apply can be viewed as a normalization of context element values. We define a transform function $T_C : C \rightarrow [0, 1]$, $c \rightarrow T_C(c)$. If we consider a context C which value is c, $T_C(c)$ is the normalized value of C. To better understand what kind of normalization we are performing, and by analogy to fuzzy logic, let us associate a predicate to each context element. T_C is going to quantify how much the value of C supports the predicate associated with C.

Table 5 shows the result of the transform for different kinds of context element. `Look Angle` is a

[5]In general, a few pairs will be retrieved at the same time. It offers the user the possibility to choose the one he considers the best.

numerical context, taking its value between 0 and 90. In this particular case, we have $T_C(c) = 1 - c/90$. For example, if `Look Angle = 0`, the context value fully supports the predicate "Look Angle is nadir", and so $T_C(0) = 1$. `Sensor Type` is a categorical context. In this case, $T_C(c)$ is one if $c =$ `Visual`, and zero otherwise. With this kind of transform, all differences between two categorical elements are considered identically (i.e., zero, the maximum of disagreement). Finally, `Desired Accuracy` is a numerical discrete context element. In this case T_C is a discrete step function (with three steps in this particular example).

Table 5: Exemple of normalization performed on three types of context element.

Context Element	Context Type	Predicate	c_1	c_2	$T_C(c_1)$	$T_C(c_2)$
Look Angle	Numeric Continuous	"Look Angle is nadir"	0^o	45^o	1	0.5
Sensor Type	Categorical	"Sensor type is Visual"	Visual	Radar	1	0
Desired Accuracy	Numerical Discrete	"Desired Acuracy is high"	middle	low	0.5	0

In the transformed context space, we use a similarity measure defined by Equation 28 to rank nearest context vectors. Normalization (N_1 and N_2 functions) is required when we want to keep the length of the vector from being taken into account in the similarity measure. This is particularly interesting for parallel vectors with different lengths. However, because vector norms and parallelism have no special meaning in our problem, we have chosen $N_1(\mathbf{u}) = N_2(\mathbf{v}) = 1$.

Thus, given the new context vector $\mathbf{C} = (c_1, .., c_n)$, and one context vector $\mathbf{C}_i = (c_{i,1}, .., c_{i,n})$ of a data base, we define the similarity S between $\mathbf{C}$ and $\mathbf{C}_i$ by the following inner product:

$$S(\mathbf{C}, \mathbf{C}_i) = \sum_{k=1}^{n} T_k(c_k) T_k(c_{i,k}) \tag{29}$$

where T_k is the transform function associated with context element k. Let n_1 be the number of parameter settings provided to the user. The n_1 highest similarity values provide the nearest context vectors present in a data base and, so, n_1 parameter settings associated with these context vectors. If we perform this ranking for every algorithm available, the n_1 overall highest similarities provide the n_1 best algorithm and parameter settings that can be proposed to the user.

As we will see in Section 9, the performance of this method is reasonably good. However, if the number of context vectors present in a data base is too large, it may take too long to use this method in an interactive scheme. Moreover, the method provides very limited learning capabilities.

Thus, in the next sections we present another method based on the notion of learning by observation, and more specifically based on conceptual clustering.

5 LEARNING BY OBSERVATION

The main goal in *learning by observation* is to build a representation of concepts supported by examples presented to the system. Generally, two problems must be addressed in learning by observation. The first one is to identify relevant concepts from the examples, and the second one is to find an appropriate representation of these concepts. Here, we describe some systems that learn by observation, and we focus on the two problems of concept formation and concept representation.

BACON [67] is a domain-specific system dedicated to chemical concept (law) formation. Concepts are represented by mathematical equations. Given a general a priori form of equation, BACON searches through the space of data, and the space of laws, to discover relations between data variables. If the way of representing concepts is natural in BACON, this is not the case for general-purpose systems of learning by observation. It is important to see that the choice of a particular representation is essential and dictates what concept a system is going to be able to learn.

The UNIMEM system [68, 69] uses a technique called Generalization Based Memory (GBM) that stores examples in a hierarchical data base (memory) describing concepts with increasing specificity. Concepts (generalizations) are represented by nodes in a discrimination net. Each node is a set of examples supporting the concept of the node. Learning is incremental. Each time a new example is considered, a controlled search is performed in the GBM to find the most specific concept(s) that describe the new example. During the search process, the matching of a new example to a generalization is based on a numerical measure of similarity between values of the new example, and values of a generalization [69].

In the BLIP system [70], both knowledge and concepts are represented by rules and predicates. To try to minimize the bias introduced by the representation mode of knowledge, knowledge rules are translated into domain-independent meta rules. These meta rules are used to generate new rules (concepts) and meta rules. The generation of new concepts is *demand driven* – that is, it is triggered by the weakness (lack of information) of the current representation of knowledge.

Both BACON and BLIP employ *constructive learning*. In the process of concept formation they create descriptors not present in the input data. This creation is very important in the learning process.

In our particular problem of finding a parameter setting for an algorithm in a given situation, we could think of expressing the expertise to determine a correct parameter setting by a set of rules as in the BLIP system – that is,

In situation 1, parameter A has to be increased by 0.1 from its standard value

However, this kind of expertise has major limitations, because many counter examples are going to be found to rules like the one above. Moreover, it is known that systems employing this kind of expertise perform poorly in *complex domains.* Studies have shaken the theory of learning involving general rule sets and imply that human expertise is based on the ability to compare a current situation to previous ones [71]. This theory is supported by neural network implementations that show that it is possible to design expert systems that do not reason (in the sense of manipulating rules) but rather act using similarity with past experiences [72].

A particular type of learning by observation is called *Learning from examples.* Both types of learning have the same goal, the determination of concept generalization. However, in the case of learning from examples, concepts are created for a particular purpose: the classification of examples. This implies new constraints for existing concepts. They have to form a partition in the space of examples – each example must verify or refine only one concept.

There are two types of learning from examples. The first one, called *concept learning*, assumes that a teacher is available to preclassify examples. The system has to produce a description of the examples of each class, which is general enough to accommodate every example, but discriminating enough to avoid interclass ambiguity. Among existing systems using the learning-from-examples approach, we cite INDUCE1.1 [73] and the system introduced by Cromwell and Kak [74]. Both systems use conjunctions of predicates to represent concepts, and both manipulate four types of data: nominal (categorical), ordinal (numerical discrete), numerical (continuous), and hierarchical. The Cromwell and Kak system includes more kinds of generalization rules, and is applied to a real case of pattern recognition in the domain of electronic component recognition.

When a system does not need preclassified examples, but rather is able to determine by itself the relevant classes (concepts) to create, we talk about *concept clustering*. The classification problem is then generally performed in two steps. The first consists in determining appropriate clusters, and the second one consists in characterizing clusters (as in concept learning). The main problem is to determine a measure to evaluate the quality of the clustering. Several criteria have been tested. Systems like CLUSTER/2 [75] and CLUSTER/S [76] use the notion of simplicity (comprehensibility) of the concept representation. Unlike statistical clustering techniques (based on numerical similarities) that produce clusters difficult to analyze, they make the assumption that the simpler the class description, the better. To avoid having a trivial partition with all examples in the same class, the final description must closely match the original examples. This clustering criterion relies on a psychological study showing that people who are asked to classify complex

data choose only one or a few simple features from the data to build a set of disjoint classes. CLUSTER/2 and CLUSTER/S use conjunction of predicates to represent concepts. In addition, CLUSTER/S uses a priori knowledge related to the application domain (expressed with a semantic net called Goal Dependency Network) to determine the relevant descriptive predicates for a given goal, and thus guide the concept formation.

Kodratoff and Tecuci [77] use a *conceptual distance* as a criterion for cluster formation. Examples are represented using conjunction of predicates. The defined conceptual distance is based on the idea that two very different examples are generalized in an expression very different from the original examples (in terms of predicates, arguments of predicates, and number of predicates in the expression), while similar examples can be generalized to themselves. Thus, both the generalization and the process of obtaining this generalization indicate the conceptual distance between examples.

A third criterion used to evaluate cluster quality consists in maximizing the inference ability of the resulting partition. The idea is that the better you can predict features based on class membership, the more advantageous it is to create such a class. Both COBWEB [78] and the system presented by Anderson [79, 80] use this kind of criterion in an incremental learning scheme. A new example is placed in the class that maximizes the predictability of the category formed by the addition of the new example, or is classified in a new category if adding the example to an existing category does not improve predictability of any of the classes. The number of classes is determined automatically by the system. These incremental schemes are interesting because learning is easily accomplished by considering new examples. However, the resulting clustering is sensitive to the order in which new examples are presented to the system. To minimize this dependency, as new examples are added, COBWEB tries to split or merge classes to improve the partitioning.

Table 6 presents a summary of important features of the learning systems described above. These features are:

- *Learning type*: Conceptual clustering is the most appropriate method for our problem. It does not require a teacher giving preclassified examples, and so reduces to a minimum the need for human expertise. It also produces a partition of the space of examples.

- *Concept formation*: similarity-based models, predictability-driven models, and maximizing comprehensibility models are fairly similar. In general, predictability-driven methods will be sensitive to the number of features shared among objects and, therefore, they tend to make the same predictions as similarity-based models [81]. The predictability-based quality measure used by COBWEB can be viewed as a continuously valued analog of the

quality measure used in CLUSTER/2 based on comprehensibility [78].

- *Representation mode*: As mentioned earlier an appropriate representation of concepts is essential to be able to express all expected concepts. Representation of concepts using conjunction of predicates is valuable for its ability to manipulate heterogeneous types of data. However, no method of conceptual clustering, using conjunction of predicates, is effective when dealing with numerical variables (in contrast with categorical ones). Hierarchical structures (trees) are also interesting because, associated with some controlled heuristic search, they provide faster methods of categorization.

- *Constructive learning*: This feature remains a challenge for most existing systems. For us, it consists in learning a new similarity relation between context elements, and ultimately new context elements.

- *Data manipulated*: As mentioned previously, we need to manipulate three kinds of data: nominal (categorical), ordinal (numerical discrete), and continuous numerical. Only learning-from-example systems, and Anderson's systems use all these types of data simultaneously.

- *Incremental learning capability*: In the domain of conceptual clustering, only Anderson's system and COBWEB provide an incremental scheme required in our problem formulation.

- *Teaching required*: Because we want to make the use of new algorithms easy, we cannot use a teacher for training. The role of a human operator using our system must only be to evaluate the algorithm results.

- *Example order and concepts are dependent*: Incremental schemes of conceptual clustering are computationally effective, but sensitive to the order in which examples are presented to the system. COBWEB uses a technique of split and merge to reduce this effect. Anderson [80] states that though the category structure can vary substantially as a function of order, the predictions delivered by the different categories do not differ much themselves.

- *Number of categories are determined automatically*: Because we have no teacher, this feature is essential.

- *Concepts produce a partition*: This feature is preferred to avoid ambiguities, but is not strictly required, if examples appear in only a few categories.

From this evaluation, we can see that the most appropriate method of learning seems to be the conceptual clustering approach using an incremental scheme of clustering and a hierarchical

representation of concepts. However, no systems having these characteristics and the ability of dealing with both categorical and numerical variables already exists. Only two systems use conceptual clustering in an incremental scheme: Anderson's system and COBWEB.

Anderson's system is not hierarchical, and has the following major inconveniences. (1) It uses a Bayesian approach relying on strong assumptions about independency of probability distributions, and form of distributions of the manipulated variables. (2) A priori knowledge that we may not have for every algorithm we use is required to set some of the several parameters of the system. (3) The number of different values of a categorical context element has to be known a priori. (4) Finally, as we will see in Section 9, parameters of the system seem to be very sensitive and hard to set correctly. Anderson's approach is concerned with deducing psychological conclusions from the parameters associated with the best (or expected) categorization – parameter values are found just by doing experiments. Furthermore, it seems that no standard values exist; parameter values are really application-dependent.

So, even if COBWEB does not deal with continuous numerical variables, it seems to be the most adapted to our problem, and it is possible to enhance COBWEB capabilities to take into account numerical variables.

6 RETRIEVAL BASED ON CONCEPTUAL CLUSTERING

Because of the number of context vectors that can be present in the data base, we cannot afford an exhaustive search for the nearest contexts. By categorizing context elements, we reduce the amount of search, because as a new context is presented to the system, it can be compared to each category instead of each context. A hierarchical organization of categories allows the search process to be even faster.

Thus, the incremental conceptual clustering scheme using a hierarchical representation of concepts seems well adapted to our problem. A conceptual clustering method of categorization can be used as a search method to categorize a new context vector **C**. The resulting category can be considered as containing the contexts nearest to **C**. The conceptual clustering quality measure after categorization of the new context vector can be used to rank the algorithms available.

As mentioned earlier, COBWEB [78] is the only system using incremental conceptual clustering that is well-adapted to our problem. It seems now necessary to present in more details COBWEB, focusing on the quality measure of clustering and the search (categorization) process of this system.

6.1 COBWEB

In COBWEB, examples are represented using attribute-value pairs. Values are only nominal or ordinal. Concept formation is based on predictability. The quality of a clustering partition $\{\omega_1, .., \omega_n\}$ is measured using the following *category utility*:

$$U = \frac{\sum_{k=1}^{n} P(\omega_k) \sum_i u_{ik}}{n} \tag{30}$$

with

$$u_{ik} = \sum_j P(C_i = V_{ij}|\omega_k)^2 - \sum_j P(C_i = V_{ij})^2 \tag{31}$$

where n denotes the number of classes in the partition. $P(C_i = V_{ij}|\omega_k)$ is the probability for context element C_i to have value V_{ij} in class ω_k, and $P(C_i = V_{ij})$ is the probability for context element C_i to have value V_{ij} over the whole partition. U measures the increase in the expected number of context element values that can be correctly guessed knowing the partition $\{\omega_1, .., \omega_n\}$ over the expected number of correct guesses when no partitioning is given [78]. As we can see, Equations 30 and 31 are fully applicable only for categorical, and eventually for ordinal values.

The search structure (called Control Structure) used to categorize a new example in COBWEB is presented in Table 7. At each level of the classification tree, the utility of the creation of a new class in the partition is compared to the one of the insertion of the new example in an existing class. The search is a recursive descent into the tree of categories. At each level, the best host for a new object is defined as the class having the maximum category utility (as calculated with Equations 30 and 31) after having added the new example. Splitting or merging is attempted on best hosts to reduce the effect of example order.

6.2 Enhancing COBWEB capabilities

Fisher's idea dictates that the category utility, expressed by Equation 31, is a measure of the increase of expectation of a value knowing a partioning, over the one when no partitioning is given. For numeric values the expectation of a value is measured by the variance of the value distribution. The narrower the distribution, the smaller the variance, and the better a value from the distribution can be predicted. Thus, for numerical values, we suggest the following term to take place in the category utility measure:

$$u_{ik} = \frac{s_i^2 - s_{i\omega_k}^2}{s_i^2 + 1} \tag{32}$$

where $s^2_{i\omega_k}$ is the variance of the distribution of values of the i'st context element in class ω_k, and s^2_i is the variance of the distribution over all the partitions. Adding one to the denominator is necessary to avoid problems with null variances. If a numerical context element is identically distributed on a class ω_k and on the whole partition, the context element is irrelevant in any partitions, and we have $s_{i\omega_k} = s_i$ and $u_{ik} = 0$

To demonstrate the categorization ability of our measure, let us consider the following simple example. Table 8 shows a categorization example with three categories and three variables. Variables are centered on the values indicated in the table. From this definition we have created 15 examples (5 from each category) by adding gaussian noise on central values. Resulting examples have been presented *randomly* for categorization, using COBWEB search structure and the category utility measure based on Equation 32. The final tree we obtain is presented in Table 9.

As we can see, the system was able to rediscover the three categories (`Class 1`, `Class 3`, and `Class 4`). Since Category 2 is nearer to Category 1 by considering the two last variables, it is comforting to see that we find this relation in the tree. In all experiments performed using this example we always found the tree shown in Table 9. Thus, the search structure, designed to minimize the effect of example order, seems to perform reasonably well even with numerical values. Results of different retrieval methods are presented in Section 9.

Since a new context has been categorized, parameters associated with nearest contexts can be retrieved. After being categorized, the new context becomes a leaf of the category tree. Near categories (brothers) in the tree represent nearest contexts from the new context.

However, the structure of the category tree can vary locally. It may be deep at some places, and not in others. As a result, the number of brothers a leaf has can be very different from place to place. Thus, the generalization may be sometimes too wide and sometimes not wide enough. To avoid this problem, and be able to always provide the same number n_1 of acceptable parameter settings, we adopt the following strategy. From the new category, we climb in the tree to pick surrounding leaf categories until a number n_0 of categories is reached ($n_0 > n_1$). Then we sort the contexts of each category depending on their similarities from the new context (nearest context first) using the similarity measure expressed in Section 4 by Equation 29. Parameter settings associated with the n_1'th first contexts are provided to the user.

This method has several advantages over the one presented in Section 4. First, it may be faster to retrieve a correct parameter setting when data bases contain a large number of examples. Second, the predictability measure performs well when lots of examples are involved. During categorization, when we approach the leaves of a tree, performance decreases. By using a

similarity measure over the n_0 nearest contexts, we greatly improve final performance. Finally, the tree structure gives very useful information about contexts and parameters, and particularly their discriminatory ability with regard to contextual situations.

7 DATA BASE UPDATE

Updating the data bases is necessary for the system to improve its performance through experience. This process is performed at the end of a session; thus, update running time is not important. During the session, update data are accumulated in a file. This file contains the successes as well as the failures of the system to provide correct parameters. Every attempt to provide a parameter vector is analyzed (see Figure 35). If the attempt was a success – that is, if the solution S returned by the process $\mathcal{A}(\mathbf{D},\mathbf{P})$ was really the one expected by the user – the automatically retrieved parameter(s), and the manually set parameters (in the case where the user had to set $\mathbf{P}$ manually), are incorporated into the data base, and associated probabilities are adjusted.

If an attempt resulted in a failure – that is, if the user had to manually indicate the solution S he wanted – then a search process is run to find the parameter vector that best reproduces the given correct solution. The context vector and the best parameter vector(s) are then added to the data base.

Any solution S is stored during a session, even if the attempt was a success. It allows the same search process to be used to learn correct parameter settings *for every algorithm available*, even if an algorithm was not chosen as appropriate. Like this, learning about all available algorithms improves at the same time.

Because the data base update is performed off-line, it can be performed continuously whenever the system is not otherwise engaged. We have not yet implemented it, but our design allows for this time to be spent finding new context elements that better resolve the selection of algorithms and parameters. This facility would constitute a very powerful capacity for the discovery of new concepts – a challenging problem in machine learning.

8 SNAKES

The automated procedure for parameter setting that we have described is, in theory, suitable for setting the parameters of virtually any algorithm. For purposes of evaluation, we have performed our experimentation using one class of feature extraction algorithms — an optimization approach

known as snakes.

Snakes were originated by Terzopoulos, Kass, and Witkin [5, 4] and have since given rise to a large body of literature. In the original implementation, the parameters were chosen interactively, and potentially had to be changed from image to image. In our own implementation [65], which is further described in the Appendix, those parameters are computed automatically and become amenable to context-specific setting.

A 2–D snake is treated as a polygonal curve C defined by a set S containing n equidistant vertices

$$S = \{(x_i\, y_i),\ i = 1, \ldots, n\}$$

that can deform itself to optimize an objective function $\mathcal{E}(C)$.

Formally, we can write the energy $\mathcal{E}(C)$ that the snake minimizes as a weighted sum of the form

$$\mathcal{E}(C) = \sum_i \lambda_i \mathcal{E}_i(C)$$

where the magnitudes of the $\mathcal{E}_i$ depend on the specific radiometry and geometry of the particular scene under consideration and are not necessarily commensurate. To determine the values of the λ_i weights in a context-specific way as opposed to an image-specific one, we have found it necessary to normalize out those influences. The dynamics of the optimization are controlled by the gradient of the objective function (Appendix, Equation a-6). We have therefore found that an effective way to achieve this result is to specify a set of normalized weights λ_i' such that

$$\sum_{1 \le i \le n} \lambda_i' = 1\ .$$

The λ_i' define the relative influences of the various components, and we use them to compute the λ_i as follows:

$$\lambda_i = \frac{\lambda_i'}{\| \vec{\nabla} \mathcal{E}_i(S^0) \|}$$

where S^0 is the estimate at the start of each optimization step. In this way we ensure that the contribution of each $\mathcal{E}_i$ term is roughly proportional to the corresponding λ_i' independently of the specific image or curve being considered.

Table 10 lists the parameters of the snake algorithm that we use to test the learning capability of our system. In practice, there are a few more, like those that define the rate of increase of the viscosity or the stopping conditions. However, since the algorithm is not very sensitive to these,

we simply fix them once and for all. The categorical parameters determine the type of snake to be used, the presence or absence of a smoothing term, the optimization procedure to be used in the absence of a smoothing term, and whether or not the endpoints of the snake ought to be fixed.

9 EXPERIMENTAL RESULTS: LEARNING AND SELECTING SNAKE PARAMETERS

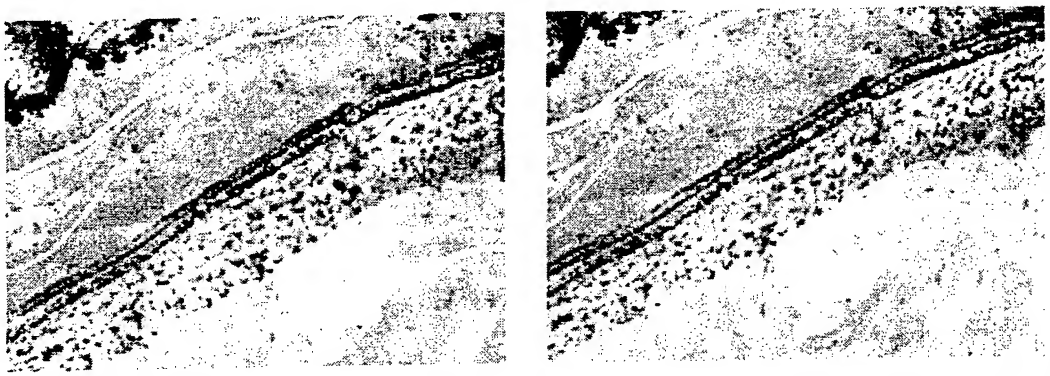
We have applied our approach to learning the snake algorithm parameters described in Table 10. Our implementation makes use of the RADIUS Common Development Environment (RCDE) [58].

9.1 Presentation

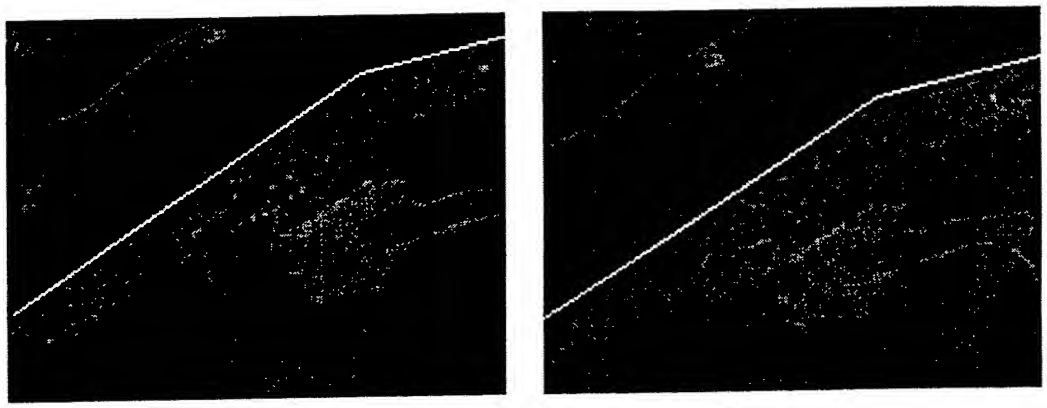
First, let us illustrate the general scheme of our system on the following example. Suppose the user's task is to delineate the ridge present in the two images depicted in Figure 36a. The user sketches the 3–D curve in the left image of Figure 36b. The camera models and digital terrain model associated with the image site are used to draw the curve in the right image of (Figure 36b).

Then the user reviews contextual information (Figure 37). There are two categories, corresponding respectively to global image context elements and curve-specific context elements. We have eight global context elements: *Look Angle* giving the look angle of the sensor, *GSD* giving the resolution, *Sensor* indicating the type of the sensor, *Element type*, *Site type*, *Season Characteristics* that point out some season particularity (snow or rain), *Illumination*, and *Sun Angle* which is important for predicting shadows. We also have eight context elements for characterizing local contextual information: *Task* indicating the class of object to be extracted, *Seed accuracy* indicating the distance between the seed curve and the expected solution, *Desired accuracy* indicating whether or not the user wants the optimized curve to strictly follow the contours of the image, *Site type*, *Material type*, *Terrain elevation*, *Seed min angle* which is the smallest computed angle between two consecutive lines defining the seed curve (minimum local curvature), and *Gradient mean* which is the average of the intensity gradient around the seed curve.

These sixteen context elements form the context vector $\mathbf{C}$. Most of the items are calculated automatically. The user can choose to not provide every item. The Parameter Selection button returns a selection of parameters based on the nearest context vector present in the data base. The user can select one of the provided parameter sets and invoke the algorithm. If the user can't find



(a)



(b)

Figure 36: (a) Two images from one site used in our tests. (b) 3–D seed curve defined in two views.

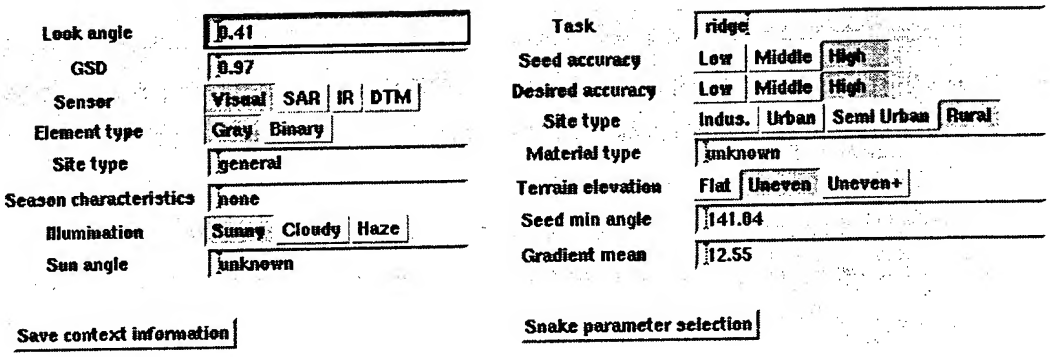


Figure 37: Context menus: global context elements (left), site-specific context elements (right).

any parameter sets giving an acceptable optimized curve, he can adjust the solution manually, set his own parameters, or modify the initial seed curve (thereby setting a new context). When the user finds an acceptable solution, the initial curve, optimized curve, and parameters are automatically saved to update the data base. The optimized curve is presented in Figure 38.

Figures 39 and 40 show subimages of the two sites we used in our tests. A site consists of several

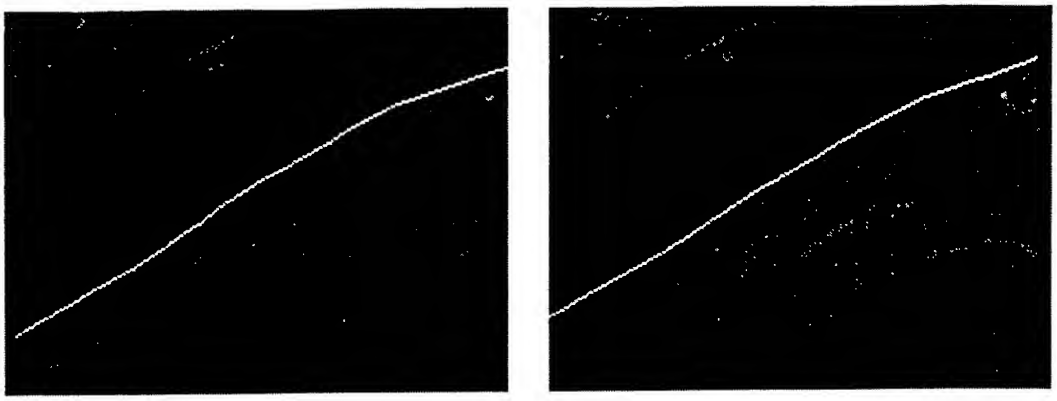
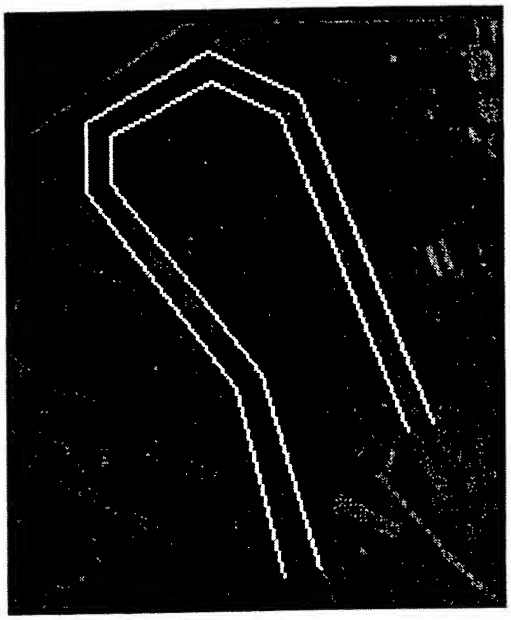


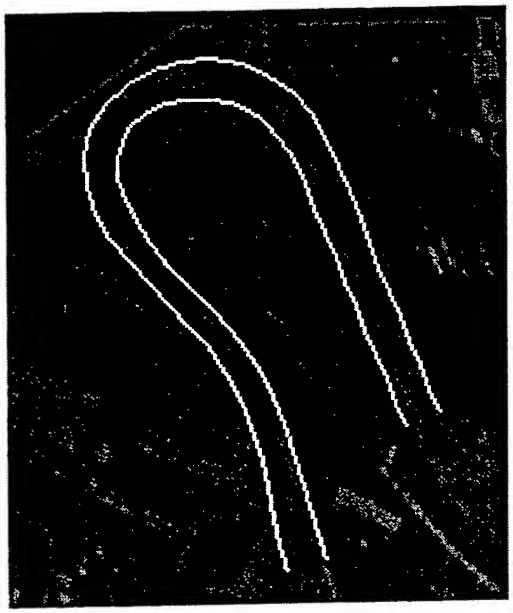
Figure 38: Snake-optimized 3–D curve

images, generally aerial images of dimensions greater than 1000 × 1000 pixels. The two test sites are very different from each other: the first one is a mountainous rural area with several industrial facilities (Figure 39a), while the second is an urban area in flat terrain (Figure 40a). Figures 39b and 40b show curves used as seeds in the snake optimization process. Figures 39c and 40c show the results of the optimization. Although the building boundaries presented in Figures 40b and c appear very similar, careful inspection will reveal that there are significant differences between the two — the optimized version is much more precise than the sketch. Finally, Figures 39d and 40d show the parameters provided by the system. The *Smoothness constraint* for the ribbon of the first site is relatively small because of the relatively high curvature of the ribbon. This aspect is captured by the context element *Seed min angle*. *Gaussian smoothing* needs to be smaller for the curve of the second site because of the relatively high edge density around the curve and, more particularly, the presence of shadow. This aspect is captured by the *Gradient mean* context element.

(a)



(b)



(c)

Parameters	Values
Type of snake	ribbon
Fixed endpoints	true
Gaussian smoothing	2
Initial step size	2.0
Stick length	10
Smoothness constraint	0.6
Width constraint	0.5
Curvature/tension ratio	1.0

(d)

Figure 39: (a) Example of images of the first test site. (b) Ribbon seed curve. (c) Snake-optimized ribbon curve. (d) Provided parameters.

Table 6: Summary of selected system comparison.

Criteria	INDUCE Michalski 80	Cromwel & Kak 91	CLUSTER/2 Michalsi, Stepp 83	CLUSTER/S Michalsi, Stepp 86	COBWEB Fisher 87	Anderson 90	Kodratoff & Tecuci 88	UNIMEM Lebowitz 83	BEACON.6 Langley et al 86	BLIP Wrobel 89
Learning type	Concept learning	Concept learning	Conceptual clustering	Conceptual clustering	Conceptual clustering	Conceptual clustering	Conceptual clustering	Concept formation	Concept formation	Concept formation
Concept formation based on	- -	- -	Maximizing Comprensibility	Maximizing Comprensibility	Predictability	Predictability	Conceptual Distance	Similarity	-	Demand Driven
Representation mode	conj. of predicates	conj. of predicates	conj. of predicates	conj. of predicates	Probabilistic tree	Categories	conj. of predicates	Discriminant net	Mathematical equations	rules
Constructive induction	limited	no	no	no	no	no	no	no	yes	yes
Data manipulated	Nominal Ordinal Hierarchical Numerical	Nominal Ordinal Hierarchical Numerical	Nominal Ordinal Hierarchical	Stuctured objects	Nominal	Nominal Numerical	Nominal Hierarchical	Numerical	Numerical	Nominal
Incremental learning capability	no	no	no	no	yes	yes	no	yes	no	yes
Teaching required	yes	yes	no	no	no	no	no	no	no	no
Example order and concepts are dependent	no	no	no	no	yes	yes	no	no	no	no
Number of categories determined automatically	no	no	no/yes[a]	no/yes[1]	yes	yes	yes	-	-	-
Concepts produce a partition	yes	yes	yes	yes	yes	yes	yes	no	-	-

[a]The number of categories is not determined automatically. However, by dividing the example classes until a manually determined bounding number of classes is attained, the system can a posteriori find what is the "best" number of classes.

Table 7: Control Structure of COBWEB (From [78])

Function COBWEB(Object, Root)
1) Update probabilities of the root
2) If Root is a leaf
THEN return the expanded leaf to accommodate the new object
ELSE find that child of Root that best hosts Object and perform one of the following
a) Consider creating a new class and do so if appropriate
b) Consider merging two best hosts and do so if appropriate and call COBWEB(Object, Merged node)
c) Consider splitting best host and do so if appropriate and call COBWEB(Object, Root)
d) call COBWEB(Object, Best child of Root)

Table 8: Example of categorization with three variables and three categories.

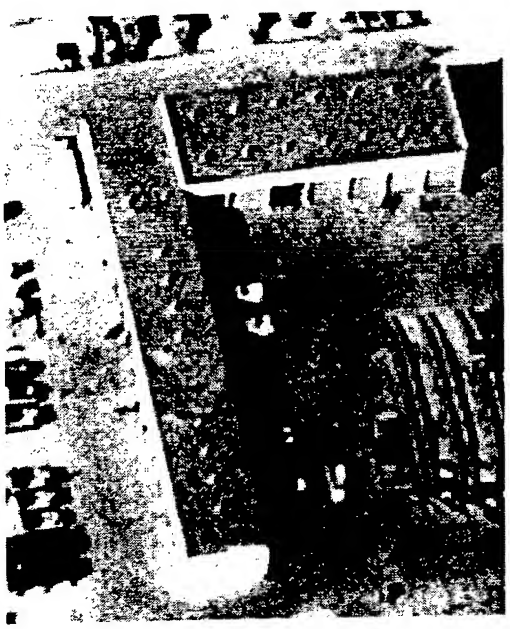
	Var1	Var2	Var3
Cat1	3	5000	60
Cat2	2	1000	100
Cat3	3	2000	150

Table 9: Example of categorization with three variables and three categories.

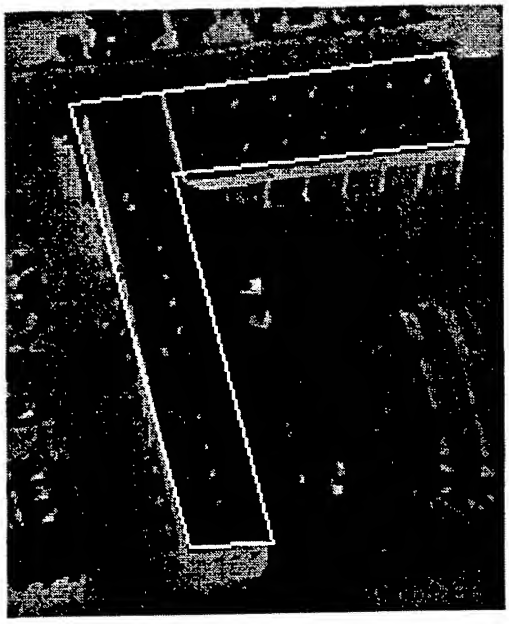
```
---> Class  0 ---> Class  2 ---> Class  4 ---> Class 25 : Ex = (4 5558 58)
                                |------> Class  8 : Ex = (3 4939 62)
                                |------> Class  7 ---> Class 16 ---> Class 18 : Ex = (3 5181 52)
                                                     |------> Class 17 : Ex = (3 5164 53)
                                          |------> Class 15 : Ex = (2 5135 48)

                    |------> Class  3 ---> Class 14 : Ex = (1 9506 100)
                               |------> Class  6 : Ex = (2 10497 103)
                               |------> Class  5 ---> Class 13 : Ex = (2 10100 106)
                                          |------> Class 12 ---> Class 24 : Ex = (2 9058 111)
                                                     |------> Class 23 : Ex = (3 9947 110)

        |------> Class  1 ---> Class 11 : Ex = (0 19230 141)
                    |------> Class 10 ---> Class 22 : Ex = (1 19718 147)
                               |------> Class 21 : Ex = (1 19810 150)
                    |------> Class  9 ---> Class 20 : Ex = (2 20439 146)
                               |------> Class 19 : Ex = (1 20618 148)
```

Table 10: Snake categorical and numerical control parameters. These parameters and related equations are defined in the Appendix.

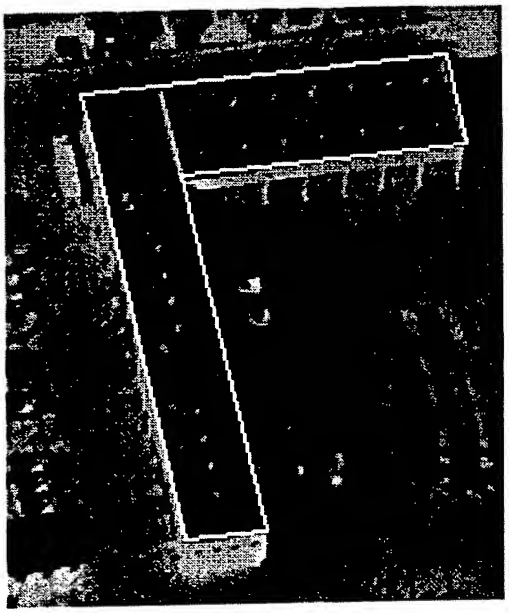
Categorical parameters	
Type of snake	Snakes can model smooth, polygonal or ribbon curves.
Fixed endpoints	The endpoints of the snake can be either fixed or not.
Numerical parameters	
Gaussian smoothing	Size of the gaussian mask used to compute image gradients
Initial step size	Δ_p, pixel step size of Equation a-7 used to compute the initial viscosity
Stick length	Initial intervertex spacing of the snake, in pixels
Smoothness constraint	λ'_D weight of the deformation component, Equation a-4
Width constraint	λ'_W weight of the width component, Equation a-9
Curvature/tension ratio	Relative contribution of tension and curvature, Equations a-4 and a-8



(a)



(b)



(c)

Parameters	Values
Type of snake	Polygonal
Fixed endpoints	not used
Gaussian smoothing	1
Initial step size	2.0
Stick length	not used
Smoothness constraint	not used
Width constraint	not used
Curvature/tension ratio	not used

(d)

Figure 40: (a) Example of images of the second test site. (b) Closed 2–D curve. (c) Snake-optimized 2–D curve. (d) Provided parameters.

9.2 System Evaluation

Testing the efficiency of our system poses three major problems with respect to maintaining test objectivity. The first one is the choice of the image curves to optimize. The snake algorithm requires some initial curves as input data. A bias in the experiments can be easily introduced by providing initial curves too close to the expected solution. In this case, the snake is going to converge toward the good solution whatever the parameters are.

The second problem is the order in which curves are optimized. Depending on the way evaluation is performed, another bias can be introduced by placing simpler examples at the end of the experiment, creating the illusion that the learning capability of the system has improved. Moreover, we have seen that the incremental scheme of retrieval is sensitive to the order of experiments. The best solution would be to define all the curves we want to test, and present them randomly to the user.

The last problem is the evaluation of the results. Some results may be acceptable for one user but not for another. This paper describes an ongoing study, and determining how well the learned result will carry over from one user to another has not yet been attempted. All the test results presented below come from a single user.

One very natural method of evaluation consists in using the system and counting the cumulative number of times the user has to adjust the result by hand. As the data base grows, the intervention required of the user should decrease, and so required adjustments should be fewer in number.

Results of this evaluation for the retrieval method based on the similarity measure expressed by Equation 29 are presented on Figure 41. We can see that for both sites, system reactivity is similar. There is a continuous decrease in the slope of each curve. This decrease is due to the effect of successfully learning suitable parameter assignments and represents an improvement in efficiency. The frequency of manual parameter setting that is required clearly decreases and tends toward zero, which is the theoretical ideal. The slope decrease also means that the user needs fewer trials to achieve his goal.

The third curve shows the hypothetical number of manual parameter settings for a user who does not use the learning module, but simply sets the parameters himself before each optimization. This curve fits the two others when the system starts to learn, and then tends to an asymptotic line with slope 2.0, indicating a mean of two manual settings per curve to optimize. The difference between the amount of hypothetical manual parameter setting and the results obtained by a user employing the learning process indicates the improvement in efficiency provided by the learning module. From the graph it is apparent that the improvement increases as the system gains

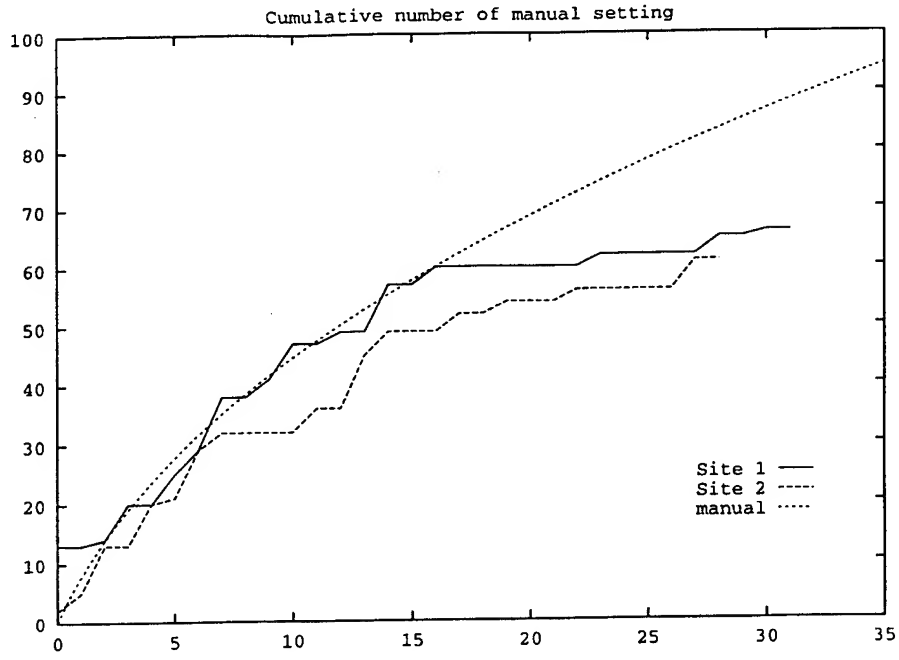


Figure 41: Cumulative number of manual settings of parameters

experience with the site.

Without the assistance of the parameter learning module, a novice user can require ten or more invocations of the snake algorithm before he attains a suitable parameter setting for each seed curve to be optimized. The capacity of the learning module to reduce the required number of invocations (to less than one per task in our experiments) represents a significant improvement in the efficiency with which snake algorithms can be employed in an interactive system.

9.3 Evaluation of Retrieval Procedure

This first experiment demonstrates clearly the learning ability of the system. However, it is sensitive to the three problems mentioned earlier: choice of original curves, order of examples, and evaluation of the results.

We therefore propose another method to evaluate more specifically the retrieval procedure. We consider the set of n curves that have already been optimized in the first experiment. For each curve $i \leq n$, we know $\mathbf{C_i}$ the contextual information associated with the curve, and $\mathbf{P_i}$ the parameter setting that has been used to find the optimized curve. Each context $\mathbf{C_i}$ is presented randomly to the system that provides a set of n_1 possible parameters. Because we know what the

correct parameter setting is ($\mathbf{P_i}$) we can measure the percentage of success of the system.

There are two ways to measure this percentage of success. The first one is to count the number of times parameter vector $\mathbf{P_i}$ belongs to the set of n_1 parameter vectors provided by the system. Because the system does not make constructive learning, and so does not propose new parameter values but instead only provides parameter vectors that have already been used, another measure is more accurate. It consists in counting the number of times parameter vector $\mathbf{P_i}$ belongs to the set of n_1 parameter vectors presented to the user when $\mathbf{P_i}$ was known to already belong to one of the categories of the system ($\mathbf{P_i}$ has already been used in a similar situation). Among the 68 optimized curves used in the experiments ($n = 68$), 39 appear to be in this situation[6].

This method is better than the first one for three reasons. First, randomly presenting the context vectors to the system is practically very easy to perform. Thus, we are going to be able to test the impact of example order on the different methods. Moreover, the user evaluation of the snake algorithm does not influence the measure that evaluates the performance of the system. Finally, the method is much less sensitive to the choice of the initial curves. Actually, this choice influences only the final structure of the category tree. "Too easy" curves tend to be associated with the same standard parameter setting, and thus always classified in the same category, thereby decreasing the number of categories and facilitating the categorization.

9.3.1 Experimental results

Table 11 shows the performance results of the three different approaches based on conceptual clustering presented in Section 6: COBWEB, Anderson's method, and our method, which use the variance for numerical variables. COBWEB uses numerical context values as categorical ones. Anderson's and our method discriminate between numerical and categorical context elements. In all experiments, unless otherwise mentioned, the number n_0 of contexts considered as the nearest context in the category structure is set to 10, and n_1, the number of parameter settings presented to the user is set to 3.

Table 11: Percentage of success of the different retrieval techniques.

	COBWEB	Anderson's	Variance method
site 1 + 2	77.6	78.9	78.1
site 1	83.9	83.3	81.7
site 2	85.8	86.5	83. 8

[6]Note that globally this number is independent of the order of the examples, even if the curves taken into account are not always the same.

Percentages presented in Table 11 are an average of success over twenty runs where examples are presented randomly to the system. As we can see, results are similar for all methods. This is explainable knowing that among the sixteen context elements used in this experiment, only two have numerical values that are different for each example (*gradient-mean* and *seed-mean-angle*). All other numerical values are associated with discrete functions or do not vary much (like *Sun angle*). These kinds of numerical values can be considered as categorical ones. Thus, the influence of the distinction between numerical and categorical values is low in this experiment. However, we can see that the system performs reasonably well, even with a small number of examples (39 examples on Site 1, 29 on Site 2).

Anderson's approach is favored by the method of parameter extraction we use after having selected the nearest contexts. Anderson's category structure is flat, and as a result, the number of context elements selected is generally greater than n_0. This is very sensitive when a small number of examples are present in the data base (as it is the case for Site 2). A priori knowledge required by the method has been set to some standard values. Other parameters required an exhaustive search among possible values to be set properly. Only one set of parameters gives correct categorization. Moreover, for the simple example presented in Section 6, the required setting was completely different, and very sensitive to the examples that were generated.

The tree structure produced by the hierarchical methods (COBWEB and Variance) are fairly similar for both methods. Analysis of normative context values[7] at the tree root shows that the classification first uses the *task* context element to categorize examples. Children of the root regroup examples with similar *task*. This is very satisfactory, because in the case of the snake algorithm, we know that the task is of first importance to determining a good parameter setting. For instance, roads and building delineation will require totally different settings.

9.3.2 Influence of example order

As mentioned earlier, incremental schemes are sensitive to the order in which examples are presented. To test this influence, we compute the standard deviation over twenty runs of the number of successes for the three retrieval techniques. Results are presented in Table 12. Again, we can see that the values are similar. They are small, too. This means that the search structure of the categorization process does not affect the inference ability of the final category structure, as pointed out by Anderson [80].

[7] Context values V_{ij} that are present in class ω_k with a conditional probability $P(A_i = V_{ij}|\omega_k)$ greater than 0.67.

Table 12: Standard deviation of the number of successes for the different retrieval techniques.

	COBWEB	Anderson's	Variance method
site 1 + 2	1.52	1.54	1.43
site 1	0.79	0.80	0.80
site 2	0.58	0.85	0.78

9.3.3 Influence of n_0 and n_1

The only two parameters of the system are n_0, the number of contexts selected as being the final nearest contexts in the category structure, and n_1, the number of parameter settings presented to the user. Figure 42 shows the influence of n_1 (with $n_0 = 10$) on the percentage of success. As we can see, the percentage of success is over 50%, even with $n_1 = 1$. It means that more than half of the time, the first guess of the system was the good one. A value of 3 seems to be reasonable for n_1.

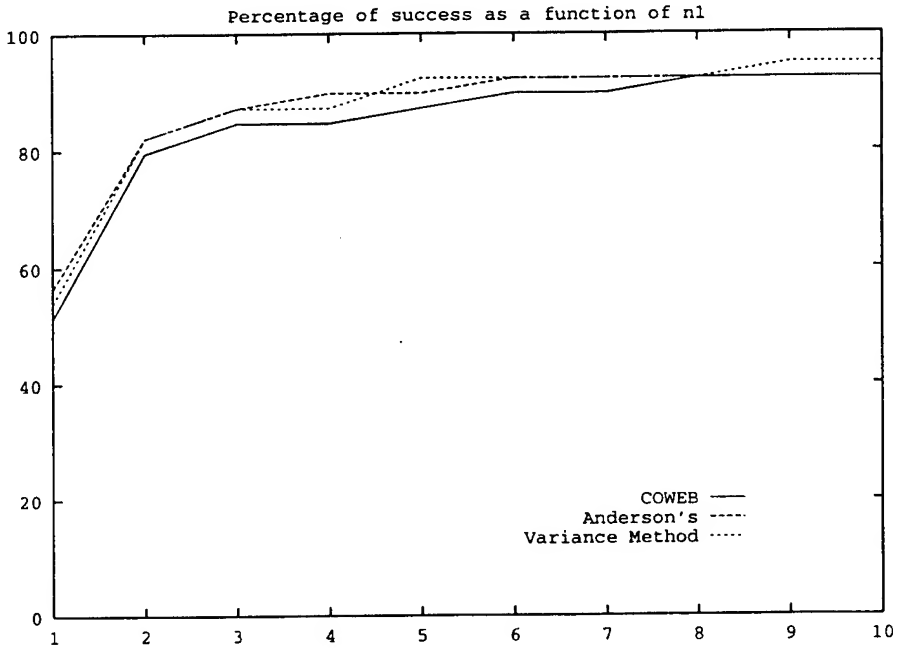


Figure 42: Influence of n_1 values over percentage of success.

As shown in Figure 43, the influence of n_0 is less important. A value $n_0 \geq 7$ is already sufficient to provide good results. It is important to note that the three methods presented use predictability to categorize a new context and similarity to retrieve n_1 parameter settings from the n_0 nearest contexts from the new context. By increasing n_0, we give more importance to the similarity measure, and less to the predictability one. When n_0 equals n, the number of examples present in

the data base, all three methods are equivalent to the one presented in Section 4 based only on similarity and with exhaustive search in the data base. As we can see in Figure 43, for $n_0 \geq 7$, performance of the variance method does not improve any more. It means that we have identical performances if the new context is categorized and then compared to only n_0 contexts, or if the new context is compared to the whole data base. This proves the ability of the heuristic search of categorization.

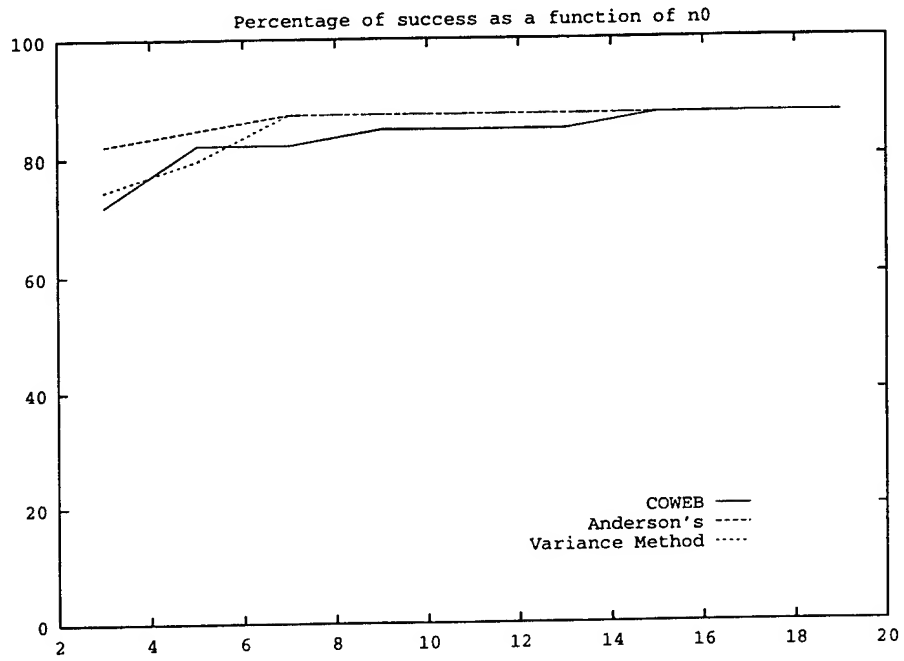


Figure 43: Influence of n_0 values over percentage of success.

9.3.4 Influence of number of context elements

We explained the similar performance of the different retrieval methods by the large number of categorical elements in the context vector we use. To draw a more accurate view of the retrieval methods, we performed tests on a subset of context elements containing three categorical, two numerical, and two ordinal context elements. These elements where chosen as the most important for categorization. Partial results we obtained are shown in Table 13. Performance of Anderson's and Variance method,s which separate categorical and numerical variables, are now better than COBWEB's method using only categorical variables.

Results of the three methods are also globally better than those shown in Figure 11. This points out an important factor : Context elements have to be adapted to the examples present in the data

Table 13: Percentage of success of the different retrieval techniques using a subset of seven context elements.

	COBWEB	Anderson's	Variance method
site 1 + 2	82.2	83.8	85.4

base. Our system has been designed for large data bases. The large number (sixteen) of context elements we use is not adapted to the small number of examples we have now in the data base. By removing context elements that are not essential because their values do not vary much in the whole data base, we improve the performance of the system. The automatic selection of context elements with regard to the number of examples already encountered is of primary interest in our future works.

Other future work remains on testing the ability of our system to choose an algorithm among several, and analyzing the various probabilities provided by the category structure. For instance, joint analysis of conditional probabilities $P(A_i = V_{ij}|\omega_k)$ and $P(\omega_k|A_i = V_{ij})$ should be an easy way to find out the discriminatory ability of each context element. This also should be a way to find a posteriori similarities between context element values. Computing this kind of similarity is essential for categorical context elements, and should greatly improve the search of similar contexts, and thus improve the retrieval performances.

10 CONCLUSION

This paper grew out of an attempt to solve a practical and important problem in interactive scene analysis: the automated selection of feature extraction algorithms and their parameters, as a function of image content and task requirements. An abstract characterization of this problem is that of mapping a multidimensional context space (representing image data and task specification) into a multidimensional algorithm-selection space (the extraction algorithms and their parameter settings). It was immediately apparent that an analytic design was infeasible. Two of the many reasons are

- We don't have effective ways of analytically describing image content.
- The range of possible tasks and image types is essentially infinite — no a priori design can hope to subsume all possible situations.

A "learning" approach appeared to be the only alternative.

The fact that the learning system is embedded in an interactive system (to deal with continuous change) offers both a challenge and an opportunity. The human operator must be aided rather than burdened by the presence of the learning system but can provide directed feedback about system performance.

Thus, the primary contribution of this paper lies in the structuring of an interactive, embedded learning system for an important problem in the design of computer vision systems — the automated selection of feature extraction algorithms and their parameters, as a function of image content, collateral data, and task requirements. The framework we have described lays the foundation for new learning mechanisms to be developed and tested — we have taken the first steps toward applying machine learning in a nonconventional learning context. We have also offered solutions to some of the subproblems that arise: how to define similarity of context vectors in which elements are both numerical and categorical, how to choose among the multiple parameter vectors that might be retrieved from the data base, and how to update the data base with experience gained through continual use of the feature extraction system. We have implemented and tested an initial design and demonstrated successful performance within a cartographic modeling domain using snake algorithms.

A SNAKES

Here we provide a mathematically precise account of the snake algorithms that we have employed within our system for learning the parameters of vision algorithms.

a.1 2–D Linear Snakes

A 2–D snake is treated as a polygonal curve C defined by a set S containing n equidistant vertices

$$S = \{(x_i\, y_i),\ i = 1, \ldots, n\} \tag{a-1}$$

that can deform itself to maximize the average edge strength along the curve $\mathcal{G}(C)$:

$$\mathcal{G}(C) = \frac{1}{|C|} \int_0^{|C|} |\nabla I(\mathbf{f}(s))|\, ds, \tag{a-2}$$

where I represents the image gray levels, s is the arc length of C, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x, y) in the image, and $|C|$ is the length of C. In practice, $\mathcal{G}(C)$ is computed by sampling the polygonal segments of the curve at regular intervals, looking up the

gradient values $|\nabla I(\mathbf{f}(s))|$ in precomputed gradient images, and summing them up. The gradient images are computed by gaussian smoothing the original image and taking the x and y derivatives to be finite differences of neighboring pixels. We have shown [65] that the points along a curve that maximizes $\mathcal{G}(C)$ are maxima of the gradient in the direction normal to the curve wherever the curvature of the curve is small. Therefore, such a curve approximates edges well except at corners. Unfortunately, $\mathcal{G}(C)$ is not convex functional and to perform the optimization, following Terzopoulos *et al.*, we minimize an energy $\mathcal{E}_(C)$ that is a weighted difference of a regularization term $\mathcal{E}_D(C)$ and of $\mathcal{G}(C)$:

$$\begin{aligned} \mathcal{E}_(C) &= \lambda_D \mathcal{E}_D(C) - \lambda_G \mathcal{G}(C) && \text{(a-3)} \\ \mathcal{E}_D(C) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 && \text{(a-4)} \end{aligned}$$

The first term of $\mathcal{E}_D$ approximates the curve's tension and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity. To perform the optimization we could use the steepest or conjugate gradient, but it would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0, && \text{(a-5)} \\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} - \frac{\partial \mathcal{G}}{\partial S}, \end{aligned}$$

where $\mathcal{E}$ is the energy of Equation a-3, α the viscosity of the medium, and S the state vector of Equation a-1 that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation a-4 is quadratic, its derivative with respect to S is linear and therefore Equation a-5 can be rewritten as

$$\begin{aligned} K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} && \text{(a-6)} \end{aligned}$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S,$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation a-6 as a set of two differential equations in the two spatial coordinates

$$\begin{aligned}(K+\alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K+\alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}}\end{aligned}$$

where K is a pentadiagonal matrix, and X and Y are the vectors of the x and y vertex coordinates. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice α is computed in the following manner. We start with an initial step size Δ_p, expressed in pixels, and use the following formula to compute the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p}\left|\frac{\partial \mathcal{E}}{\partial S}\right|, \tag{a-7}$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the non linear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

The snakes described above have proved very effective at modeling smooth curves. Some objects, however, such as buildings, are best modeled as polygons with sharp corners. They can be handled in this context by completely turning off the smoothness term. Such objects typically have a relatively small number of corners, and the optimization is performed using a standard optimization technique.

a.2 3–D Linear Snakes

Snakes can be naturally extented to three dimensions by redefining C as a 3–D curve with n equidistant vertices $S = \{(x_i\ y_i\ z_i)\},\ i = 1,\ldots,n\}$ and considering its projections in a number of

images for which we have accurate camera models. The average edge strength $\mathcal{G}(C)$ of Equation a-2 becomes the sum of the average edge strengths along the projection of the curve in the images under consideration, and the regularization term of Equation a-4 becomes

$$\begin{aligned} \mathcal{E}_D(C) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 + (2z_i - z_{i-1} - z_{i+1})^2 \end{aligned} \tag{a-8}$$

Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation a-6 as a set of three differential equations in the three spatial coordinates:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)Z_t &= \alpha Z_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Z}\right|_{Z_{t-1}} \end{aligned}$$

where X,Y, and Z are the vectors of the x,y, and z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{G}(C)$ is computed by summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation a-7.

a.3 Ribbons

2–D snakes can also be extended to describe ribbon-like objects such as roads in aerial images. A ribbon snake is implemented as a polygonal curve forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The state vector S becomes the vector $S = \{(x_i\ y_i\ w_i)\},\ i = 1,\ldots,n\}$ and the average edge strength the sum of the edge strengths along the two boundary curves. Since the width of roads tends to vary gradually, we add an additional energy term of the form

$$\mathcal{E}_W(C) = \sum_i (w_i - w_{i-1})^2 \tag{a-9}$$

$$\Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} = LW,$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}_(C) = \lambda_D \mathcal{E}_D(C) + \lambda_W \mathcal{E}_W(C) - \lambda_G \mathcal{G}(C)$$

and at each iteration the system must solve the three differential equations:

$$\begin{aligned}
(K+\alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\
(K+\alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\
(K+\alpha I)W_t &= \alpha W_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial W}\right|_{W_{t-1}}
\end{aligned}$$

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector $S = \{(x_i\ y_i\ z_i\ w_i)\},\ i = 1,\ldots,n\}$ and at each iteration the system must solve four differential equations, one for each coordinate.

REFERENCES

[1] D. Gerson, "RADIUS: The Government Viewpoint," in *ARPA Image Understanding Workshop*, Jan. 1992.

[2] J. Mundy and P. Vrobel, "The Role of IU Technology in RADIUS Phase II," in *Unpublished*, May 1994.

[3] T. M. Strat and W. D. Climenson, "RADIUS: Site Model Content," in *ARPA Workshop on Image Understanding*, Nov. 1994.

[4] D. Terzopoulos, A. Witkin, and M. Kass, "Symmetry-seeking Models and 3D Object Reconstruction," *International Journal of Computer Vision*, vol. 1, pp. 211–221, 1987.

[5] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[6] P. Fua and Y. G. Leclerc, "Model Driven Edge Detection," *Machine Vision and Applications*, vol. 3, pp. 45–56, 1990.

[7] P. Fua and C. Brechbuhler, "Imposing Hard Constraints on Soft Snakes," in *European Conference on Computer Vision*, (Cambridge, England), pp. 495–506, April 1996. Available as Tech Note 553, Artificial Intelligence Center, SRI International.

[8] R. C. Jain and T. O. Binford, "Ignorance, Myopia, and Naiveté in Computer Vision Systems," *CVGIP: Image Understanding*, vol. 53, pp. 112–117, Jan. 1991.

[9] Y. Aloimonos, "Purposive and Qualitative Action Vision," in *Proc. DARPA Image Understanding Workshop*, (Pittsburgh, PA), pp. 816–828, September 1990.

[10] V. Clément, G. Giraudon, S. Houzelle, and F. Sandakly, "Interpretation of Remotely Sensed Images in a Context of Multisensor Fusion using a Multispecialist Architecture," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 31, July 1989.

[11] D. D. Fu, K. J. Hammond, and M. J. Swain, "Vision and Navigation in Man-Made Environments: Looking for Syrup in all the Right Places," in *Proceedings of the Workshop on Visual Behaviors*, (Seattle, WA), pp. 20–26, June 1994.

[12] T. M. Strat and M. A. Fischler, "Context-Based Vision: Recognizing Objects Using Both 2D and 3D Imagery," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 1050–1065, Oct. 1991.

[13] T. M. Strat and M. A. Fischler, "The Role of Context in Computer Vision," in *ICCV Workshop on Context-Based Vision*, (Cambridge, MA), June 1995.

[14] J. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs, "The RADIUS Common Development Environment," in *DARPA Image Understanding Workshop*, (San Diego,CA), pp. 215–226, Morgan Kaufmann, 1992.

[15] P. Fua and P. Sander, "Segmenting Unstructured 3D Points into Surfaces," in *European Conference on Computer Vision*, (Genoa, Italy), pp. 676–680, April 1992.

[16] R. Szeliski and D. Tonnesen, "Surface Modeling with Oriented Particle Systems," in *Computer Graphics, SIGGRAPH Proceedings*, vol. 26, pp. 185–194, July 1992.

[17] L. Quam, "Hierarchical Warp Stereo," in *DARPA Image Understanding Workshop*, pp. 149–155, 1984.

[18] H. Nishihara, "Practical Real-Time Imaging Stereo Matcher," *Optical Engineering*, vol. 23, no. 5, 1984.

[19] T. Kanade and M. Okutomi, "A Stereo Matching Algorithm with an Adaptative Window: Theory and Experiment," in *DARPA Image Understanding Workshop*, Morgan Kaufmann, September 1990.

[20] E. P. Baltsavias, *Multiphoto Geometrically Constrained Matching*. PhD thesis, Institute for Geodesy and Photgrammetry, ETH Zurich, December 1991.

[21] F. Devernay and O. D. Faugeras, "Computing Differential Properties of 3–D Shapes from Stereoscopic Images without 3–D Models," in *Conference on Computer Vision and Pattern Recognition*, (Seattle, WA), pp. 208–213, June 1994.

[22] B. Wrobel, "The evolution of Digital Photogrammetry from Analytical Photogrammetry," *Photogrammetric Record*, vol. 13, pp. 765–776, April 1991.

[23] C. Heipke, "Integration of Digital Image Matching and Multi Image Shape From Shading," in *International Society for Photogrammetry and Remote Sensing*, (Washington, D.C.), pp. 832–841, 1992.

[24] Y. G. Leclerc and A. F. Bobick, "The Direct Computation of Height from Shading," in *Conference on Computer Vision and Pattern Recognition*, (Lahaina, Maui, Hawaii), June 1991.

[25] P. Fua and Y. G. Leclerc, "Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading," *International Journal of Computer Vision*, vol. 16, pp. 35–56, September 1995.

[26] R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 2nd ed., 1987. A Wiley-Interscience Publication.

[27] P. Gill, W. Murray, and M. Wright, *Practical Optimization*. London a.o.: Academic Press, 1981.

[28] D. Metaxas and D. Terzopoulos, "Shape and Nonrigid Motion Estimation through Physics-Based Synthesis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 6, pp. 580–591, 1991.

[29] A. Amini, S. Tehrani, and T. Weymouth, "Using Dynamic Programming for Minimizing the Energy of Active Contours in the Presence of Hard Constraints," in *International Conference on Computer Vision*, pp. 95–99, 1988.

[30] C. Brechbühler, G. Gerig, and O. Kübler, "Parametrization of Closed Surfaces for 3-D Shape Description," *Computer Vision and Image Understanding*, vol. 61, pp. 154–170, March 1995.

[31] Rosen, "Gradient Projection Method for Nonlinear Programming," *SIAM Journal of Applied Mathematics*, vol. 8, pp. 181–217, 1961.

[32] L. Quam, "Road Tracking and Anomaly Detection," in *DARPA Image Understanding Workshop*, pp. 51–55, Morgan Kaufmann, May 1978.

[33] P. Fua, "Parametric Models are Versatile: The Case of Model Based Optimization," in *ISPRS WG III/2 Joint Workshop*, (Stockholm, Sweden), September 1995.

[34] W. Neuenschwander, P. Fua, G. Székely, and O. Kubler, "Making Snakes Converge from Minimal Initialization," in *DARPA Image Understanding Workshop*, (Monterey, CA), Morgan Kaufmann, November 1994.

[35] D. Terzopoulos, A. Witkin, and M. Kass, "Symmetry-seeking Models for 3D Object Reconstruction," *International Journal of Computer Vision*, vol. 1, pp. 211–221, Oct. 1987.

[36] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[37] L. Cohen, "On Active Contour Models and Balloons," *Computer Vision, Graphics, and Image Processing: Image Understanding*, vol. 53, pp. 211–218, March 1991.

[38] L. Staib and J. Duncan, "Boundary Finding with Parametrically Deformable Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 1061–1075, Nov. 1992.

[39] F. Leymarie and M. Levine, "Tracking deformable objects in the plane using an active contour model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 617–634, June 1993.

[40] D. Terzopoulos and R. Szeliski, "Tracking with Kalman Snakes," in *Active Vision* (A. Blake and A. Yuille, eds.), The MIT Press, 1992.

[41] B. Bascle and R. Deriche, "Stereo Matching, Reconstruction und Refinement of 3D Curves Using Deformable Contours," in *International Conference on Computer Vision*, (Berlin, Germany), pp. 421–430, 1993.

[42] M. Berger and R. Mohr, "Towards Autonomy in Active Contour Models," in *Tenth International Conference on Pattern Recognition*, (Atlantic City, NJ), pp. 847–851, June 1990.

[43] O. Henricsson and W. Neuenschwander, "Controlling Growing Snakes by Using Key-Points," in *International Conference on Pattern Recognition*, (Jerusalem, Israel), pp. 68–73, Oct. 1994.

[44] D. Terzopoulos, "On matching deformable models to images," *Topical Meeting on Machine Vision Tech. Digest Series*, vol. 12, pp. 160–167, 1987.

[45] R. Courant and D. Hilbert, *Methods of mathematical physics*, vol. 1. Wiley: New York, 1989.

[46] V. Arnold, *Ordinary Differential Equations*. MIT Press, 1973.

[47] A. W. Bush, *Perturbation methods for engineers and scientists*. CRC Press, 1992.

[48] W. Neuenschwander, *Elastic deformable contour and surface models for 2-D and 3-D image segmentation*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, Sept. 1995.

[49] M.-O. Berger, *Les contours actifs: modélisation, comportement et convergence*. PhD thesis, Instiut National Polytechnique de Lorraine, Inria Lorraine, Centre de Recherche en Informatique de Nancy, France, Feb. 1991.

[50] C. A. Davatzikos and J. L. Prince, "Adaptive active contour algorithms for extracting and mapping thick curves," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 524–529, 1993.

[51] T. Strat and M. Fischler, "Natural Object Recognition: A Theoretical Framework and Its Implementation," *Proc. IJCAI-91*, August 1991.

[52] T. M. Strat, *Natural Object Recognition*. Springer-Verlag, New York, 1992.

[53] T. M. Strat, "Employing Contextual Information in Computer Vision," in *DARPA Workshop on Image Understanding*, pp. 217–229, April 1993.

[54] L. Quam and T. Strat, "SRI Image Understanding Research in Cartographic Feature Extraction," in *International Society for Photogrammetry and Remote Sensing*, (Munich), Sept. 1991.

[55] P. Graham, *On Lisp: Advanced Techniques for Common Lisp*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[56] M. Fischler and H. Wolf, "Linear Delineation," in *Conference on Computer Vision and Pattern Recognition*, pp. 351–356, June 1983.

[57] A. J. Hanson and L. Quam, "Overview of the sri cartographic modeling environment," in *DARPA Workshop on Image Understanding*, pp. 576–582, Apr. 1988.

[58] J. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs, "The RADIUS Common Development Environment," in *Proc. of AIPR, Washington, DC*, Oct. 1991. Also *in Proc. of DARPA Image Understanding Workshop*, San Diego, California, 1992.

[59] T. M. Strat and M. A. Fischler, "Context-based vision: Recognizing objects using both 2d and 3d imagery," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 1050–1065, Oct. 1991.

[60] B. Draper, A. R. Hanson, and E. Riseman, "Learning knowledge-directed visual strategies," in *DARPA Workshop on Image Understanding*, pp. 933–940, 1992.

[61] B. Draper, A. R. Hanson, and E. Riseman, "Statistical properties of learning recognition strategies," in *DARPA Workshop on Image Understanding*, pp. 557,565, 1993.

[62] D. M. McKeown, W. A. Harvey, and J. McDermott, "Rule-Based Interpretation of Aerial Imagery," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 570–585, Sept. 1985.

[63] B. Draper, R. Collins, J. Brolio, A. R. Hanson, and E. Riseman, "The Schema System," *International Journal of Computer Vision*, vol. 3, no. 2, pp. 209–250, 1989.

[64] V. Clement, G. Giraudon, S. Houzelle, and F. Sandakly, "Interpretation of remotely sensed images in a context of multi sensor fusion using a multi-specialist architecture," *IEEE Trans. on Geoscience and Remote Sensing*, 1993.

[65] P. Fua and Y. G. Leclerc, "Model Driven Edge Detection," in *DARPA Image Understanding Workshop*, (Cambridge, Massachusetts), pp. 1016–1021, April 1988.

[66] Z. Wang, S. Wong, and Y. Yao, "An analysis of vector space models based on computational geometry," in *ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 152–160, June 1992.

[67] P. Langley, J. Zytkow, H. Simon, and G. Bradshaw, *The search for Regularity: Four Aspects of Scientific Discovery*, pp. 425,469. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[68] M. Lebowitz, *Concept Learning in a Rich Input Domain*, pp. 193–214. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[69] M. Lebowitz, *Experiments with Incremental Concept Formation: UNIMEM*, pp. 103,138. Kluwer Academic Publishers, Boston, 1987.

[70] S. Wrobel, *Demand-driven Concept Formation*, pp. 289,319. Springer Verlag, 1989.

[71] H. Dreyfus, "La portee philosophique du connexionisme," in *Introduction aux sciences cognitives*, In French, Editions Folio, 1987.

[72] A. Giacometti, *Modeles hybrides de l'expertise*. PhD dissertation, Telecom Paris. In French, 1993.

[73] R. Michalski, "Pattern recognition as rule-guided inductive inference," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 349,361, July 1980.

[74] R. Cromwell and A. Kak, "Automatic generation of object class descriptions using symbolic learning techniques," in *Proc. of the ninth National Conference on Artificial Intelligence*, pp. 710,717, 1991.

[75] R. Michalski and R. Stepp, "Automated construction of classifications: Conceptual clustering versus numerical taxonomy," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, pp. 396,409, July 1983.

[76] R. Stepp and R. Michalski, *Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects*, pp. 471,498. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[77] Y. Kodratoff and G. Tecuci, "Learning based on conceptual distance," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 10, pp. 897,909, Nov. 1988.

[78] D. Fisher, *Knowledge Acquisition Via Incremental Conceptual Clustering*, pp. 139,172. Kluwer Academic Publishers, Boston, 1987.

[79] J. Anderson, *The adaptive character of thought.* Hillsdale, NJ: Erlbaum, 1990.

[80] J. Anderson, "The adaptative nature of human categorization," *Psychological Review*, vol. 18, pp. 409–429, 1991.

[81] W. Ahn and D. Medin, "A two-stage model of category construction," *Cognitive Science*, vol. 16, pp. 81,121, 1992.